НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено
Завідувач кафедри
_____ Олександр Коваль
(підпис)
“___”_____2020р.

# ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

з напряму підготовки 122 Комп’ютерні науки та інформаційні технології

на тему Автоматизація тестування знань з дисципліни «Лінгвістичне забезпечення САПР»

Виконав: студент 4 курсу, групи ТР-62

Кардашов Олександр Вадимович _____ _____
(прізвище, ім’я, по батькові) (підпис)

Керівник доц., к. т. н., доц., Стативка Ю. І. _____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Консультант _____ _____ _____
(назва розділу) (вчені ступінь та звання, прізвище, ініціали) (підпис)

Рецензент _____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

**Національний технічний університет України**
**“Київський політехнічний інститут імені Ігоря Сікорського”**

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки 122 Комп’ютерні науки та інформаційні технології

Спеціалізація Геометричне моделювання в інформаційних системах

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр Коваль
(підпис)
”___”_____2020р.

# ЗАВДАННЯ
**на дипломну роботу студенту**

_____Кардашову Олександру Вадимовичу_____
(прізвище, ім’я, по батькові)

1. Тема роботи ___Автоматизація тестування знань з дисципліни_____ «Лінгвістичне забезпечення САПР»_____
_____

керівник роботи__Стативка Юрій Іванович, доцент, кандидат технічних наук_____
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”25” травня 2020р. № 1268-с

2. Строк подання студентом роботи _____
3. Вихідні дані до роботи __мова програмування Python, фреймворк web2py_____
_____

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)__проаналізувати обраний фреймворк та програмне забезпечення для розробки, проаналізувати розроблену базу даних системи тестування студентів, впровадити систему реєстрації та аутентифікації користувачів системи, розробити за запрограмувати алгоритм проходження тестування користувачами, розробити та запрограмувати алгоритм адміністрування наповнення змістом системи_____
_____
_____
_____

5. Перелік ілюстративного матеріалу

Автоматизація тестування знань з дисципліни «Лінгвістичне забезпечення САПР», Завдання, web2py. Шаблон Model View Controller, Варіанти використання системи користувачем-студентом, Варіанти використання системи користувачем-викладачем, Варіанти використання системи редактором тестів, Модель бази даних, Інтерфейс реєстрації та аутентифікації, Інтерфейс користувача, Сторінка введення та редагування граматики Бекуса-Наура, Інтерфейс редагування особистого аккаунта студента, Інтерфейс редагування наповнення системи, Інтерфейс історії відповідей користувачів.

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| | | | |

7. Дата видачі завдання ”16”__грудня__2019 р.

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів виконання дипломної роботи | Термін виконання етапів роботи | Примітки |
|---|---|---|---|
| 1. | Затвердження теми роботи | 01.10.2019 р. | |
| 2. | Вивчення та аналіз задачі | 15.11.2019 р. | |
| 3. | Розробка архітектури та загальної структури системи | 15.01.2020 р. | |
| 4. | Розробка структур окремих підсистем | 15.02.2020 р. | |
| 5. | Програмна реалізація системи | 15.03.2020 р. | |
| 6. | Оформлення пояснювальної записки | 25.05.2020 р. | |
| 7. | Захист програмного продукту | 11.05.2020 р. | |
| 8. | Передзахист | 10.06.2020 р. | |
| 9. | Захист | 18.06.2020 р. | |

Студент _____ _____Кардашов О. В._____
(підпис) (прізвище та ініціали,)

Керівник роботи _____ _____Стативка Ю. І._____
(підпис) (прізвище та ініціали,)

# АНОТАЦІЯ

Записка містить 53 сторінки, 34 рисунки, 3 додатки та 6 посилань.

Мета роботи – створити автоматизовану систему тестування студентів, яка надавала б дистанційний доступ до тестувань студентів-користувачів, так і доступ адміністраторів до редагування контенту системи – набору та складу тестів.

Вибір фреймворку web2py для роботи, було зроблено, оскільки даний фреймворк дозволяє розробляти динамічні кросплатформні веб-додатки використовуючи вбудовані до фреймворку модулі.

У результаті було розроблено систему тестування студентів з гнучким набором завдань, які можна персоналізувати під особисті потреби користувачів та адміністраторів системи.

Ключові слова: фреймворк web2py, кросплатформний веб-додаток, система тестування, система аутентифікації, редагування системи.

# ABSTRACT

The note contains 51 pages, 33 figures, 3 appendices and 2 links.

The purpose of the work is to create an automated system of student testing, which would provide remote access to testing of student users, as well as access of administrators to edit the content of the system - the set and composition of tests.

The choice of the web2py framework for operation was made for use in the development process, because this framework allows you to develop dynamic cross-platform web applications using built-in modules.

As a result, a system of student testing was developed with a flexible set of tasks that can be personalized to the personal needs of users and system administrators.

Keywords: web2py framework, crossplatform web application, testing system, authentication system, system editing.

# ЗМІСТ

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

SQL – декларативна мова програмування для взаємодії користувача з базою даних.

HTML (Hyper Text Markup Language) – стандартизована мова розмітки веб-сторінок мережі Інтернет.

XML (eXtendable Markup Language) – мова розмітки документів.

CSS (Cascading Style Sheets) – формальна мова опису стилів документів, що написані мовою розмітки.

URL (Uniform Resource Locator) – система уніфікованих адрес електронних ресурсів.

MVC (Model View Controller) – схема розподілення бази даних додатку, інтерфейсу та керуючої логіки на окремі компоненти.

IP-адреса (Internet Protocol Address) – унікальна мережева адреса вузла в комп’ютерній мережі.

# ВСТУП

На сьогоднішній день, під час активного розвитку інформаційних технологій викладачі перш за все потребують впровадження автоматизації певних процесів та систем. Автоматизовані системи дозволяють зрозуміло впорядкувати навчальний процес у вишах — структурувати час співробітників для викладення матеріалу, створити контрольований автоматизований розклад для викладачів і студентів, впровадити системи контролю знань студентів та перевірки усіх необхідних контрольних робіт, курсових проектів, тощо.

Серед великого переліку таких процесів можна виокремити автоматизацію проходження перевірки контролю знань студентів під час проходження курсу з дисципліни «Лінгвістичне забезпечення САПР» у вищому навчальному закладі. Метою розробки системи є надання викладачеві змоги автоматизувати майже весь процес проходження тестування студентів, а отже надання змогу оптимізувати витрату часу викладача на перевірку завдань. Водночас, система повинна надавати змогу студенту отримати інтуїтивно зрозумілу та чесну систему оцінювання своїх знань, яка дає змогу моніторити власні знання, більш того, система повинна надавати змогу проходження тестування дистанційно, без обов’язкової присутності студента в аудиторії.

Впровадження такої системи дозволить підвищити мобільність навчального процесу і дозволить вчасно контролювати рівень знань студентів без повного переривання навчального процесу, що особливо актуально під час неможливості забезпечення стаціонарного проходження процесу навчання.

Така автоматизована система тестування студентів може застосовуватися у будь-яких установах, що включають у процес своєї діяльності контроль знань, наприклад, у школах, університетах, на різноманітних підприємствах, що зацікавлені у моніторингу кваліфікації своїх працівників. Тести можуть мати будь-який вигляд – від розгорнутих відповідей до класичних тестів з наданими варіантами відповідей.

Втручання для зміни системи під конкретні вимоги замовника повинні бути мінімальними.

Для виконання поставлених задач необхідно розробити:

- алгоритм системи аутентифікації користувача;
- алгоритм системи проходження тестування студентом;
- алгоритм редагування наповнення системи адміністратором;
- алгоритм функції перегляду історії тестувань;
- базу даних системи;

# 1 АВТОМАТИЗАЦІЯ ПІДГОТОВКИ ТА ПРОХОДЖЕННЯ ТЕСТІВ

## 1.1 Мета розробки

Метою роботи є автоматизація підготовки та виконання студентами тестових завдань з дисципліни «Лінгвістичне забезпечення САПР».

Викладач повинен отримати можливість автоматизувати процес перевірки правильних відповідей студентів для оптимізації навчального процесу та уникнення людського фактору у даному процесі. Процес перегляду тестів повинен супроводжуватися можливістю фільтрації отриманих даних за самими тестуваннями, за датою проходження тесту, логіном студента, правильними та неправильними відповідями. Також викладач повинен отримати можливість адмініструвати систему шляхом додавання, редагування та видалення тестувань для забезпечення актуальності навчальної програми.

Студентові необхідно надати можливість реєстрації та подальшої аутентифікації у персональному аккаунті системи. Під час збереження результатів тестувань студентів, система повинна надати викладачу можливість перевіряти дату та час проходження тестування студентом. Після проходження тестування студент повинен отримати інформацію щодо правильності або неправильності зроблених відповідей, проте, без безпосередніх підказок.

Така система повинна надавати дистанційний доступ студентам до проходження тестувань та викладачам до їх адміністрування, що забезпечить мобільність та стабільність навчального процесу у виші.

## 1.2 Загальна структура системи

Взявши за основу теоретичні відомості та програмні матеріали, створити автоматизовану систему тестування студентів на базі web-додатку, що складається з наступних модулів:

- користувацький інтерфейс;
- панель адміністратора;

Користувацький інтерфейс повинен бути інтуїтивно зрозумілим та забезпечувати персональний доступ користувачів до системи тестування шляхом реєстрації персональних аккаунтів та подальшій авторизації користувачів. Реєстраційні дані користувачів повинні складатися з:

- логіну користувача;
- паролю користувача;
- прізвища та імені;
- електронної пошти користувача;

Система повинна надавати користувачу повний перелік доступних тестів, після чого користувач повинен мати змогу обрати необхідний тест і заповнити його. Після того як користувач заповнить тест, дані про пройдене тестування треба відправити на серверну частину системи для збереження у вигляді таблиці бази даних і подальшої перевірки результатів тестування викладачем. Система повинна надати користувачу результати тестування відразу після відправки ним даних на сервер. Результати надають користувачу інформацію про вірну або невірну відповідь на кожний пункт пройденого ним тесту. Після перегляду результатів тестування користувач повинен повернутися на головну сторінку з переліком тестів.

Панель адміністратора повинна надавати адміністратору системи – тобто викладачу або лаборанту можливість як переглядати результати тестування студентів так і наповнювати базу даних тестами, при цьому обмежувати їх від можливості вивести систему з ладу шляхом внесення невиправних змін в створену базу даних.

Отже головна сторінка панелі адміністратора повинна містити дві основні функції:

- доступ до перегляду результатів тестування студентів;
- доступ до редагування тестів;

На сторінці з результатами тестування студентів необхідно надати можливість адміністратору відсортувати за зростанням отримані дані таблиці з відповідями за усіма наявними у таблиці полями а також відфільтрувати загальний список пройдених тестів за:

- датою проходження;
- ім’ям користувача;
- назвою тесту;
- вірними або невірними відповідями;

Сторінка редагування тестів повинна надавати можливість адміністратору переглядати наявні тести, змінювати назву тестів, видаляти тести з бази даних, та додавати нові тести до бази даних. Усі зміни, що вносяться адміністратором у список тестів необхідно враховувати у історії збережених даних з відповідями студентів для того щоб при перегляді цих даних зберігалася актуальність наявної інформації. Для того щоб убезпечити адміністратора від ненавмисного видалення тесту або зміни його назви, що внесе невиправні зміни до історії відповідей студентів, необхідно вивести на екран форму підтвердження видалення тесту з бази даних. Додавання тесту до бази даних відбувається шляхом заповнення загальної форми тестів.

Сторінка перегляду обраного тесту адміністратором повинна складатися з таблиці у якій відображатимуться завдання та правильні відповіді на завдання. Функціонал сторінки перегляду тесту повинен надавати наступні можливості:

- перегляд тесту у вигляді таблиці;
- редагування кожного завдання тесту;
- видалення завдання тесту;
- додавання нового завдання;

Як і у випадку з видаленням таблиці тесту або зміною назви, видалення завдання теж повинно супроводжуватись підтвердженням дії адміністратором на сторінці з відповідною формою та відповідними змінами що синхронно зі змінами у тесті будуть вноситися до таблиці з відповідями студентів.

Для роботи над створенням програмного коду web-додатку було вирішено використовувати web-орієнтований фреймворк web2py на базі мови програмування Python. Для розмітки сторінок web-додатку використовується мова HTML. За замовчуванням фреймворком використовується бібліотека підтримки мови SQL – SQLite.

# 2 ЗАСОБИ ДЛЯ РОЗРОБКИ СИСТЕМИ АВТОМАТИЗАЦІЇ ПІДГОТОВКИ ТА ПРОХОДЖЕННЯ ТЕСТІВ

## 2.1 Фреймворк web2py

Для створення такого web-додатку використовується фреймворк web2py, що базується на мові програмування Python. Web2py підтримує HTTP, HTTPS-запити та HTTP, HTTPS-відповіді, cookies, сесії. Фреймворк забезпечує механізмами аутентифікації і контролю доступу за ролями. Весь процес розробки, відладки, тестування віддаленої бази даних може здійснюватися, з використанням браузера, через веб-інтерфейс [1].

Основою архітектури додатків, що використовують web2py є шаблон проектування Model-View-Controller (MVC) [1]. Models, Views, Controller є частинами будь-якого додатку web2py (Рисунок 2.1).

Диспетчер направляє URL-адресу, на яку був запит, на виклик функції в контролері. Функція може повертати строку або словник символів – хеш-таблицю. Дані у словнику відображаються шаблоном представлення – View. Якщо користувач робить запит на HTML-сторінку, то словник відображає дані що зберігає на HTML-сторінці. Якщо користувач робить запит на одну і ту саму сторінку в XML, то web2py намагається знайти відповідний файл шаблону View, який може відображати словник в XML [3]. Розробник може створювати представлення View для візуалізації сторінок у будь-якому з протоколів, які підтримує фреймворк web2py, а саме: HTML, XML, JSON, RSS, CSV.

Сервер може бути як локальним web2py-сервером, там і стороннім сервером. Пунктирні стрілки відображають зв’язки з однією або декількома базами даних. Запити до бази даних можуть бути написані як на мові SQL так і з використанням класу DAL, а отже, додатки web2py не залежать від однієї системи управління базами даних.

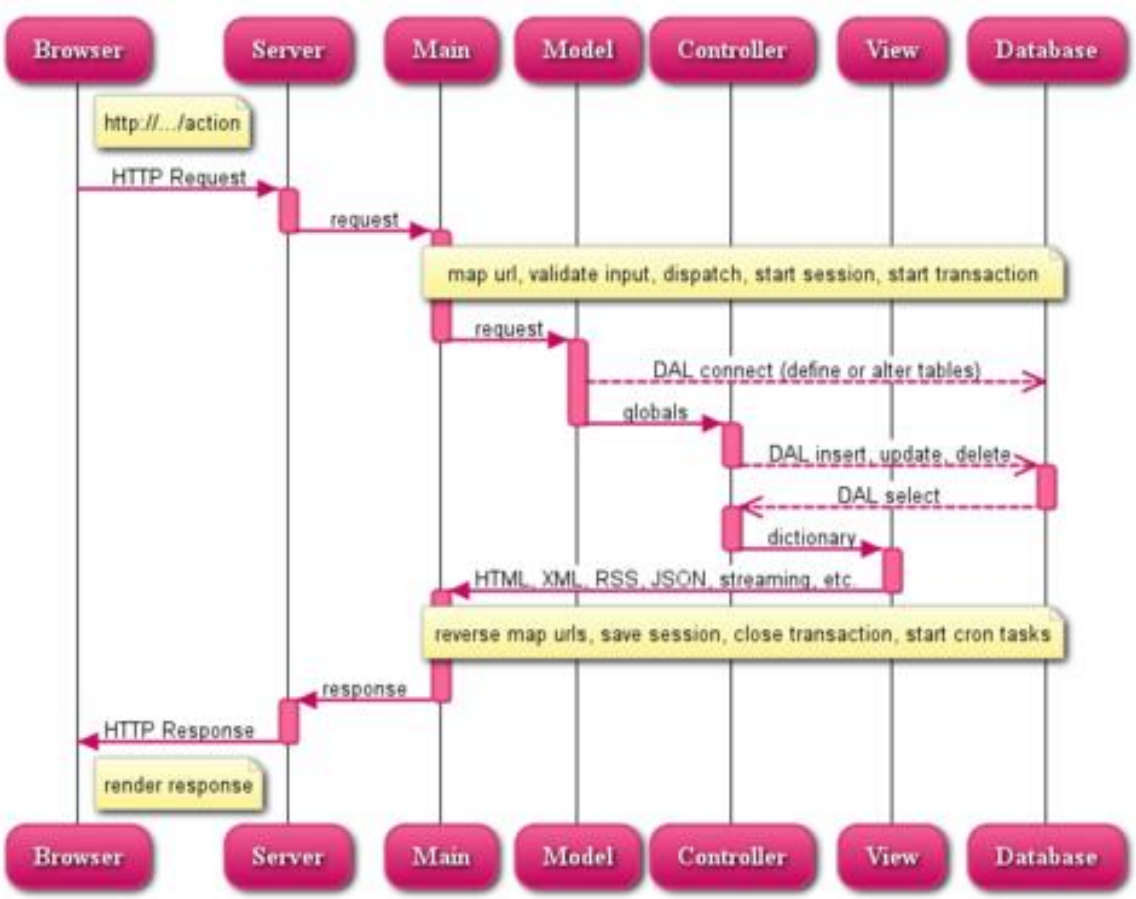



Рисунок 2.1 – Схема проходження запиту-відповіді використовуючи модель MVC [1]

У Models відбувається підключення додатку до бази даних та оголошення таблиць і їх властивостей. Також даний модуль дозволяє імпортувати до системи та інтергувати вбудовані стуктури баз даних, на кшталт системи авторизації користувачів Auth.

Файл Controller зберігає код з методами – для окремої сторінки пишеться окремий метод. У цьому модулі зберігається основна логіка алгоритмів системи. Відбувається обробка пост-запитів веб-сторінок системи та формування форм та структур даних що повертають методи. Під час роботи над поставленим завданням неодноразово було використано таблиці та форми, формування яких відбувалося не за допомогою мови тегів HTML, а за допомогою вбудованих методів фреймворку web2py, що є аналогічними інтерпретаціями тегів, але дозволяють створювати заздалегідь стуктурований набір даних для якого можна окремо створити алгоритм поведінки при виведенні на веб-сторінці. Для форм створюється логіка дій при «прийнятті» даних, що були введені користувачем. Такі структури даних дуже зручно

використовувати, оскільки при побудові веб-сторінки можна суттєво покращити оптимізацію коду та уникнути небажаних повторень одного і того самого коду форм у різних сценаріях поведінки алгоритму або при різних діях користувача системи.

Модуль View викликається тоді, коли спрацював виклик методу з відповідним сторінці ім'ям у файлі Controller. Метою шаблону View є візуалізація в HTML-код форм та структур даних у словнику, що повертається методом з файлу Controller. Може існувати окремий файл модулю View для кожного методу з файлу Controller, так і один файл. У файлі View відбувається побудова основної структури веб-сторінок системи – таблиць, розмітки, відбувається розміщення засобів керування систему для користувача а також тут, в основному, відбувається формування пост-запитів що використовуються для формування списку фільтрів даних, сортування та для окремих елементів логіки алгоритму функціонування веб-сторінок системи.

Якщо виникає помилка під час розробки за допомогою фреймворку web2py, вона записується в логи і розробнику видається повідомлення для можливості відслідковувати помилки [1]. Помилки та вихідний код доступні лише адміністратору, котрий може шукати їх за датою та IP-адреси клієнта [1].

Фреймворк має вбудований графічний інтерфейс роботи з базами даних, що суттєво спрощує роботу з ними. Web2py включає рівень абстракції баз даних (Data Access Level, DAL), що динамічно генерує SQL-запити і виконує їх на багатьох СКБД без необхідності написання їх мовою SQL, але, за потреби, SQL-запити можна виконувати напряму. Взаємодія з базою даних додатку відбувається шляхом виконання SQL-команд реляційної СКБД SQLite, що підтримується фреймворком web2py. Під час роботи з SQLite необхідно враховувати, що дана СКБД не може змінювати таблицю та тип стовбця, а просто зберігає нові значення відповідно до нового типу. Редагувати зміст таблиць, або відслідковувати внесені зміни до бази даних додатку web2py можна використовуючи веб-інтерфейс appadmin. Фреймворк забезпечує автоматичну міграцію баз даних, тобто, якщо оголошення таблиці у коді буде змінено, то web2py змінить цю таблицю автоматично, відповідно до внесених змін. Міграцію даних можна за потреби заборонити для окремої таблиці. Процес міграції також записується у логи, документуючи внесені зміни.

## 2.2 Бібліотека SQLite

SQLite – це бібліотека, що реалізує автономний, безсерверний, з нульовою конфігурацією, транзакційний двигун (engine) бази даних SQL. Бібліотека є найбільш широко розгорнутою базою даних з відкритим доступом для використання у проектах.

Двигун SQLite є вбудованим двигуном бази даних SQL. На відмінну від більшості інших баз даних SQL, у SQLite немає окремого серверного процесу. SQLite виконує зчитування та запис даних безпосередньо в звичайні файли на диску. Повна база даних SQL з кількома таблицями, індексами, тригерами та представленнями зберігається лише в одному файлі. Формат файлу бази даних є кросплатформенним – а отже його можна вільно копіювати між 32-бітовою та 64-бітовою системами або між архітектурами big-endian та little-endian. Ці функції надають SQLite вагомих переваг у виборі способу збереження бази даних. Розмір бібліотеки SQLite може бути менше ніж 600КБ, залежно від цільової платформи та налаштувань оптимізації компілятора [2]. 64-розрядний код більший, а деякі функції оптимізації компілятора, такі як жорстке вбудовування функцій та циклічне розгортання, можуть призвести до того, що об’єктний код буде набагато більшим [2]. Проте існує компроміс між використанням пам’яті та швидкістю. SQLite зазвичай працює швидше, якщо виділити більше оперативної пам’яті. Тим не менш, продуктивність, як правило, досить хороша навіть в умовах малого об’єму виділеної оперативної пам’яті. Залежно від способу його використання, SQLite може бути швидшим, ніж прямий ввід-вивід файлової системи.

## 2.3 Бібліотека pyDAL

pyDAL - це чистий рівень абстракції бази даних Python.

pyDAL динамічно генерує SQL-команди у режимі реального часу, використовуючи вказаний діалект для бек-енду бази даних, тож не доводиться писати

SQL-код або знати різноманітні діалекти SQL. Код, що використовує pyDAL можна портувати серед на інші типи баз даних.

pyDAL походить від оригінальної DAL веб-сторінки web2py з метою широкої сумісності. pyDAL не вимагає web2py і може використовуватися в будь-якому контексті Python.

# 3 ВАРІАНТИ ВИКОРИСТАННЯ СИСТЕМИ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ

Web-додаток, що реалізує автоматизовану систему оцінювання студентів складається з двох основних модулів – користувацького інтерфейсу та інтерфейсу адміністратора системи. Обидва модулі функціонують незалежно один від одного з точки зору користувача модуля, проте тісно пов’язані між собою зв’язками бази даних.

Взаємодія з автоматизованою системою оцінювання студентів може відбуватися з точи зору трьох ролей – студента, викладача, адміністратора системи.

Опис взаємодії користувачів з системою подається у вигляді діаграм прецедентів, які складаються з акторів та прецедентів, що відображають варіанти використання системи [4].

## 3.1 Функції студента у системі

Актор «Студент» має можливість реєстрації у системі тестування та подальшої авторизації у ній. Система надає можливість обрати потрібний тест з наданого списку. Проходження тесту відбувається на окремій сторінці з відповідною формою. Після підтвердження відправлення форми актор «Студент» може переглянути результати тестування. На веб-сторінку виводиться список завдань пройденого студентом тесту і навпроти кожного виводиться, чи правильну відповідь дав студент. Після перегляду даних відповідей на завдання тесту студент має змогу повернутися за головну сторінку системи для подальшого вибору необхідних тестів (Рисунок 3.1).

Студент має можливість змінити пароль в будь-який час роботи з системою або вийти зі свого аккаунту у системі.

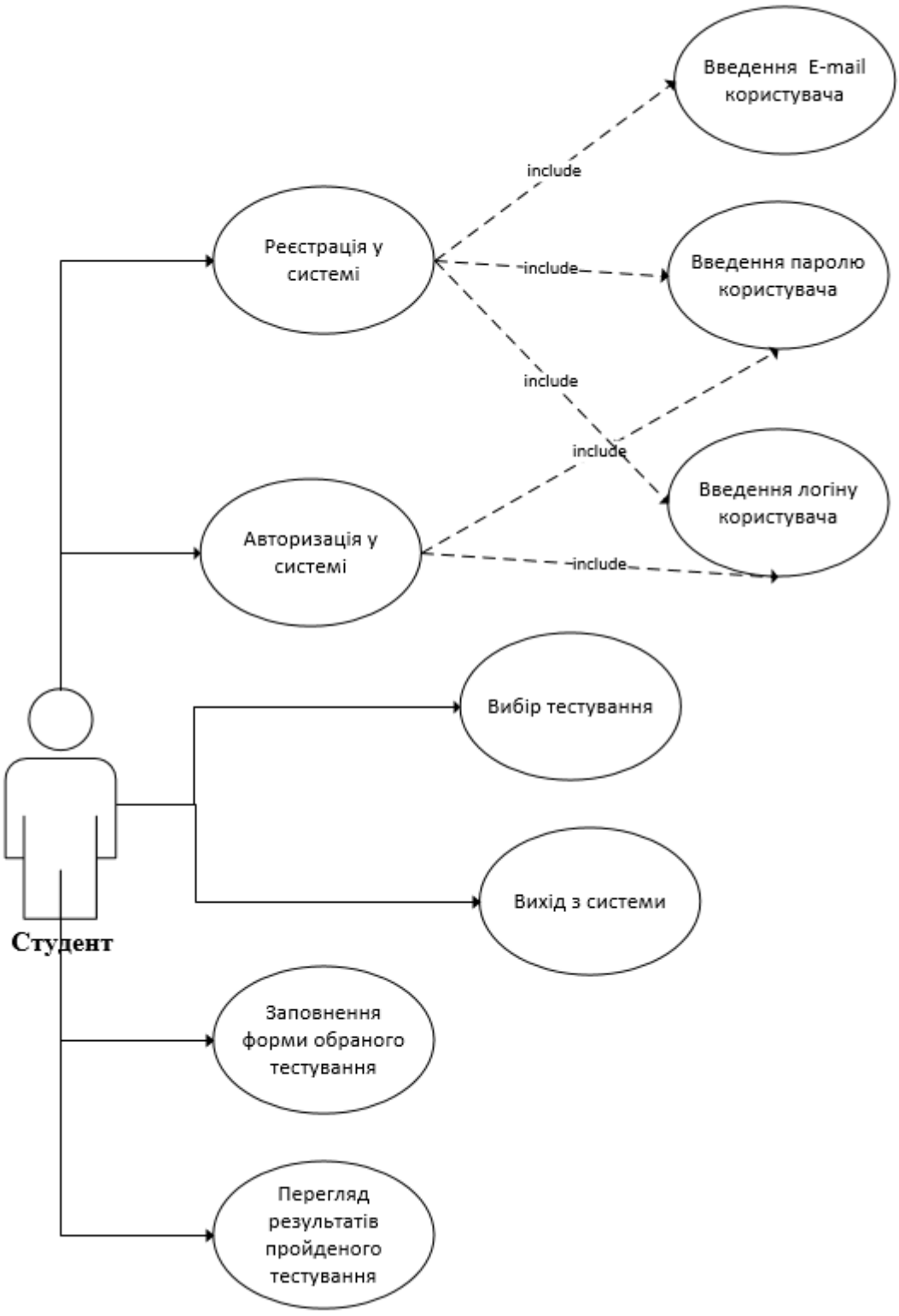



Рисунок 3.1 - Use-case діаграма інтерфейсу користувача «Студент»

## 3.2 Функції викладача у системі

Актор «Викладач» має можливість перегляду загальної таблиці результатів тестування та вибору даних з таблиці да допомогою фільтрів за датою додавання результатів тестування студента до таблиці результатів, за номером тесту та за іменем студента. Також «Викладачу» надається можливість сортувати таблицю з відповідями студентів за усіма параметрами що є у таблиці (Рисунок 3.2).

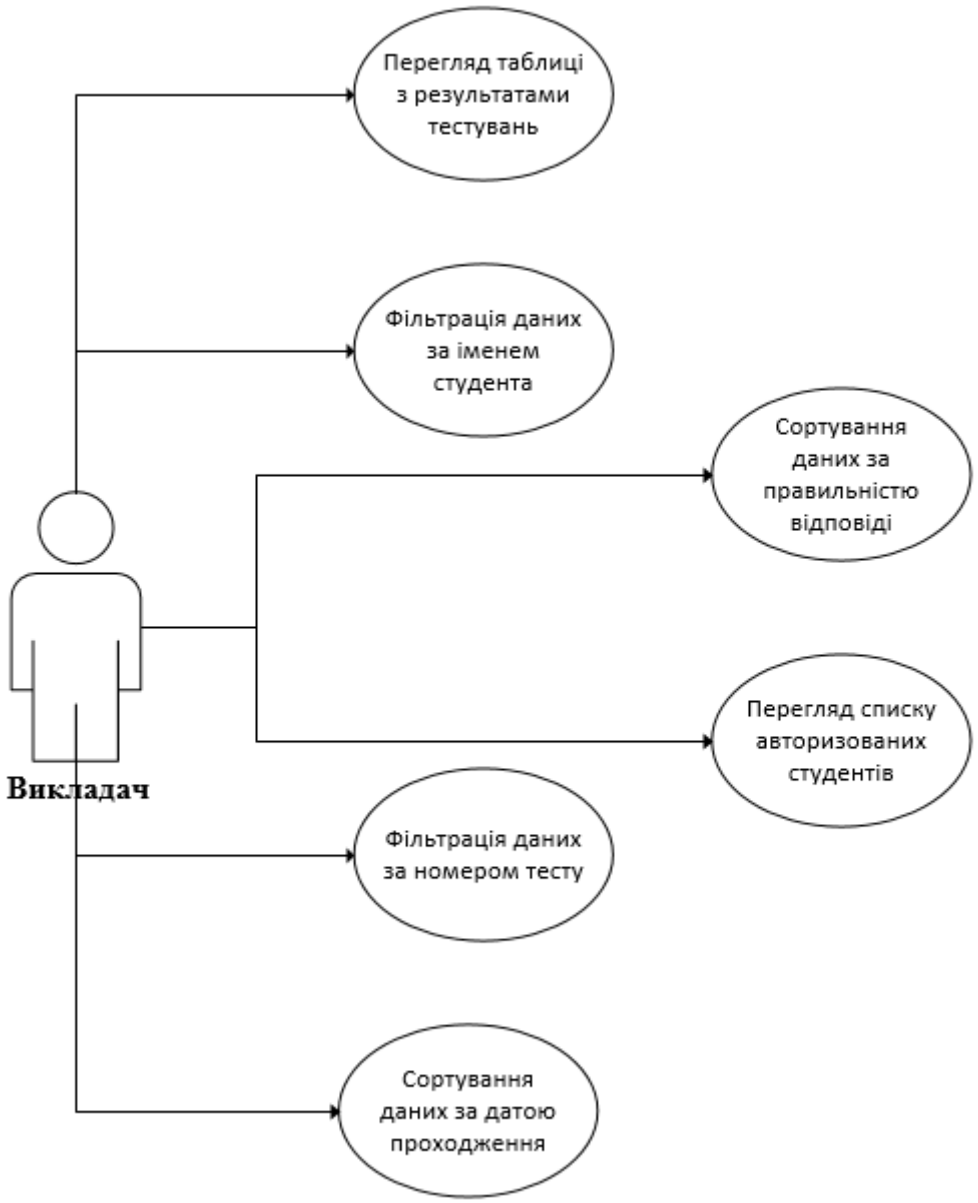



Рисунок 3.2 - Use-case діаграма інтерфейсу «Викладача»

Функція сортування даних надає можливість «Викладачу» впорядкувати дані таблиці за алфавітним порядком, якщо мова йде про текстову колонку таблиці, наприклад, назва тесту, або за порядком зростання чисел, якщо йдеться про числові дані, наприклад, ідентифікатори, або за плином дати, від пізнішої до найближчої, або за зростанням часу.

## 3.3 Функції редактора тестів

Акторові «редактор тестів» система надає дві основні групи дій – редагування переліку тестів та безпосередньо складу тестів. «Редактор тестів» виконує функцію редактора контенту системи.

До першої групи можливостей «Редактора тестів» можна віднести перегляд списку тестів. У список входять і ті тести, що не були заповнені, а були лише оголошені. «Редактор тестів» має можливість відредагувати назву вже існуючого тесту а також створити новий тест. Обидві функції вимагають лише введення назви тесту, а заповнення тесту завданнями відбувається за допомогою інших модулів. Відбувається так зване оголошення тесту. Також надається можливість видаляти тест з переліку. Як при видаленні, так і при редагуванні назви тесту адміністратору необхідно підтвердити свою дію для уникнення небажаних незворотних змін до змісту історії відповідей студентів, що зберігає тести, та, як наслідок їх назви. Надається, також, можливість встановити та відредагувати «дедлайн» для кожного тесту.

До другої групи можливостей «Редактора тестів» відносяться дії з наповненням тестів. Актор має можливість додавати нові завдання до тесту, а також редагувати їх. Після вибору відповідної функції адміністратору необхідно буде заповнити поля завдання тесту та правильної відповіді. Як і у випадку з видаленням тесту, для функцій редагування та видалення завдання, потребується підтвердження дії від адміністратора (Рисунок 3.3).

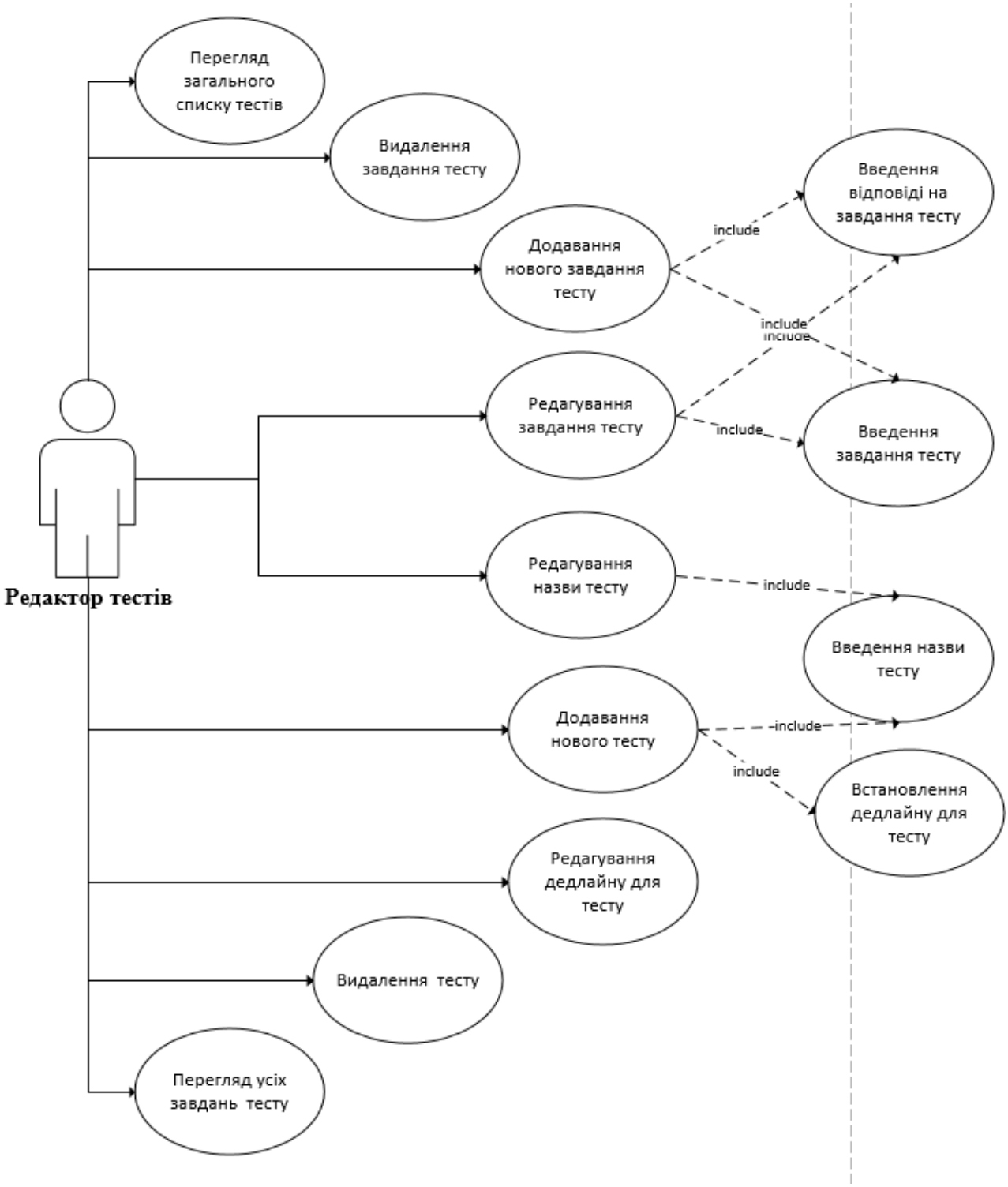



Рисунок 3.3 - Use-case діаграма інтерфейсу «Редактор тестів»

# 4 МОДЕЛЬ БАЗИ ДАНИХ

## 4.1 Концептуальна модель бази даних

Концептуальна модель бази даних є описом основних таблиць і зв'язків між ними (Рисунок 4.1). Концептуальна модель будується на основі предметної області, для якої створюється база даних [5].


Рисунок 4.1 - Концептуальна модель бази даних

Обраний тип бази даних системи – реляційна модель бази даних. У реляційній моделі дані групуються за зберігаються у вигляді таблиць що мають стовпці та рядки. Кажен стовпець реляційної бази даних зберігає однорідні дані, що об’єднуються за певним параметром, де кожен елемент даних має однакову природу. Отже, навіть різні типи даних, можуть зберігатися в одному стовпці, наприклад, дата та текстова строка, якщо обидва записи пов’язані логічною суттю [6].

Web2py має вбудовані засоби аутентифікації користувачів, що і було використано при розробці даної системи. Аутентифікація web2py з точки зору схеми бази даних це група таблиць “auth”.

Центральна таблиця групи “auth” – це auth_user. Саме її поля заповнюються при реєстрації користувача у системі. Її первинний ключ id виступає ідентифікатором користувача, та цим полем вона пов’язана з такими таблицями як: auth_cas, auth_event, auth_membership, userAnswer.

Web2py також дає можливість аутентифікації за ролями, і у базі даних це відображається таблицею auth_group. Таким механізмом можна надавати різний рівень доступу по системи, наприклад, для студента, викладача та адміністратора. Таблиця auth_group пов’язана своїм первинним ключем id з таблицями auth_membership та auth_permission.

Оскільки таблиця userAnswer зберігає інформацію про відповіді користувачів на тести, то вона пов’язана з таблицею tasks за полем id_expression таблиці tasks, що зберігає інформацію про завдання тестів, та з таблицею tests за полем name_table_expression що містить перелік створених тестів. Таблиця userAnswer має інформацію про подію аутентифікації користувача у системі, а отже пов’язана з таблицею auth_event за полем id_authEvent. Також за полем id_user можна отримати доступ до інформації про зареєстрованого користувача що надав відповідь.

## 4.2 Призначення таблиць бази даних

Таблиця auth_user зберігає імена користувачів та їх прізвища, адреси електронної пошти та паролі.

Таблиця auth_group зберігає групи або ролі для користувачів у структури «багато-до-багатьох». За замовчуванням кожен користувач знаходиться у своїх групі, але користувач може знаходитися в декількох групах, як і кожна група може мати декілька користувачів. Група ідентифікується через роль і опис призначення.

Таблиця auth_membership пов’язує користувачів і групи у структури «багато-до-багатьох».

Таблиця auth_permission пов’язує групи і дозволи для групи. Дозвіл ідентифікується за іменем і, необов’язково, за ідентифікатором таблиці і записом. Наприклад, члени певної групи можуть бути дозвіл на “update” деякого запису таблиці.

Таблиця auth_event зберігає інформацію про події аутентифікації, реєстрації у системи, створення або видалення групи користувачів, виходу користувача з аккаунту системи. При подіях, пов’язаними з користувачем, також зберігає IP-адресу користувача. Виконує роль своєрідного журналу подій.

Таблиця auth_cas використовується для Центральної служби аутентифікації (Central Authentication Service, CAS).

Таблиця userAnswer зберігає інформацію про кожну відповідь на завдання тестів кожного студента. Призначена для як для зберігання історії відповідей, так і для збереження інформації про дату та час занесення відповіді до бази даних. Крім того зберігає інформацію про те, чи є відповідь користувача на питання тесту правильною.

Таблиця tests зберігає інформацію про створені тести системи. З полів має лише ім’я та ідентифікатор.

Таблиця tasks зберігає інформацію про усі завдання кожного тесту. Завдання ідентифікується за допомогою назви тесту та ідентифікатора. Кожен запис зберігає разом з виразом завдання правильну відповідь на завдання.

## 4.3 Параметри та поля таблиць бази даних системи

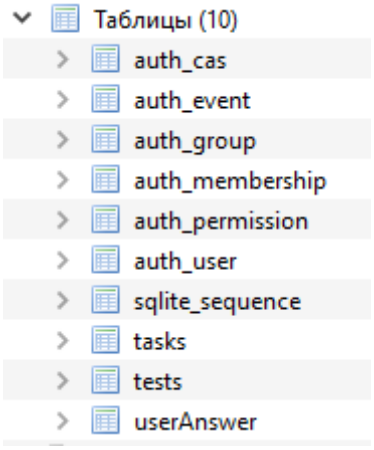



Рисунок 4.2 - Перелік таблиць бази даних системи

База даних складається з 10 таблиць (Рисунок 4.2). При створенні таблиці у web2py автоматично генерується автоінкрементне поле id, що є первинним ключем. Кожна таблиця містить первинний ключ (Primary key), за допомогою якого відбувається явна ідентифікація та доступ до конкретного запису кожної таблиці.

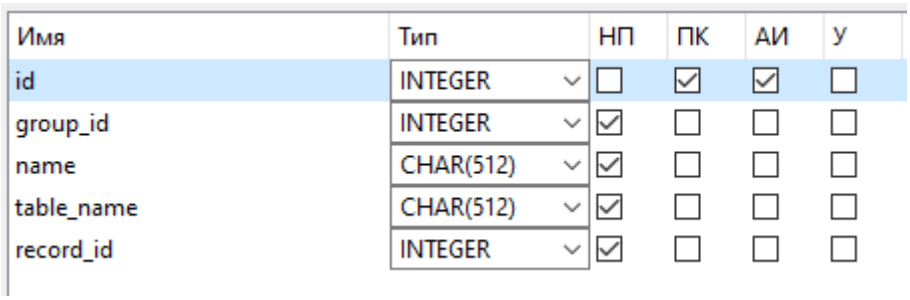
Нижче наведено опис параметрів таблиць бази даних системи:

| Имя | Тип | НП | ПК | АИ | У |
|---|---|---|---|---|---|
| id | INTEGER | ☐ | ☑ | ☑ | ☐ |
| first_name | CHAR(128) | ☑ | ☐ | ☐ | ☐ |
| last_name | CHAR(128) | ☑ | ☐ | ☐ | ☐ |
| email | CHAR(512) | ☑ | ☐ | ☐ | ☐ |
| password | CHAR(512) | ☑ | ☐ | ☐ | ☐ |
| registration_key | CHAR(512) | ☐ | ☐ | ☐ | ☐ |
| reset_password_key | CHAR(512) | ☐ | ☐ | ☐ | ☐ |
| registration_id | CHAR(512) | ☐ | ☐ | ☐ | ☐ |
| username | CHAR(128) | ☑ | ☐ | ☐ | ☐ |

Рисунок 4.3 - Параметри таблиці “auth_user”

Таблиця “auth_user” має 5 необхідних для заповнення поля при створенні запису, а саме: first_name, last_name, email, password, username (Рисунок 4.3). Якщо користувач при заповненні реєстраційної форми залишить будь-яке з цих полів пустим, то система повідомить його про це відповідним надписом про необхідність

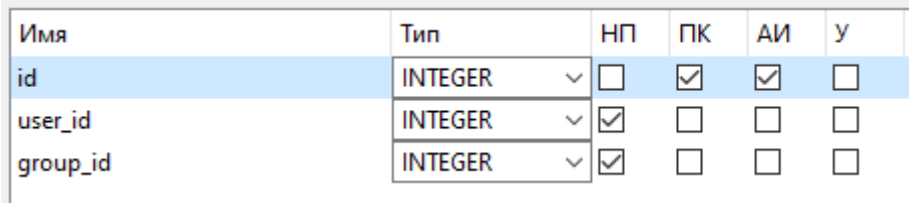
заповнити пропущені поля. Поле id виступає первинним ключем з функцією автоінкременту.



| Имя | Тип | НП | ПК | АИ | У |
|---|---|---|---|---|---|
| id | INTEGER | ☐ | ☑ | ☑ | ☐ |
| group_id | INTEGER | ☑ | ☐ | ☐ | ☐ |
| name | CHAR(512) | ☑ | ☐ | ☐ | ☐ |
| table_name | CHAR(512) | ☑ | ☐ | ☐ | ☐ |
| record_id | INTEGER | ☑ | ☐ | ☐ | ☐ |

Рисунок 4.4 - Параметри таблиці “auth_permission”

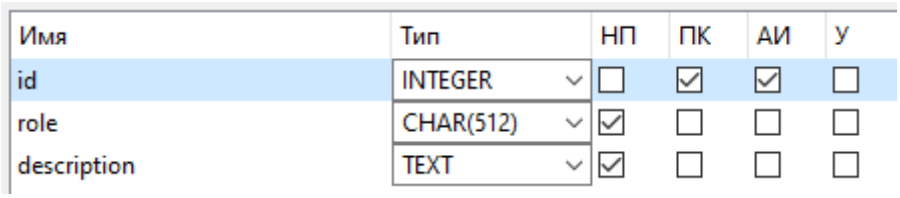
Таблиця “auth_permission” має 4 необхідних для заповнення поля: group_id, name, table_name, record_id (Рисунок 4.4). Поле group_id зберігає ідентифікатор групи, для якої прописано дозвіл. Поле name зберігає назву запису дозволу для групи користувачів. Поле table_name зберігає назву групи. Поле record_id зберігає ідентифікатор запису для групи, якщо певний дозвіл для групи користувачів не один. Поле id - первинний ключем з автоінкрементом.



| Имя | Тип | НП | ПК | АИ | У |
|---|---|---|---|---|---|
| id | INTEGER | ☐ | ☑ | ☑ | ☐ |
| user_id | INTEGER | ☑ | ☐ | ☐ | ☐ |
| group_id | INTEGER | ☑ | ☐ | ☐ | ☐ |

Рисунок 4.5 - Параметри таблиці “auth_membership”

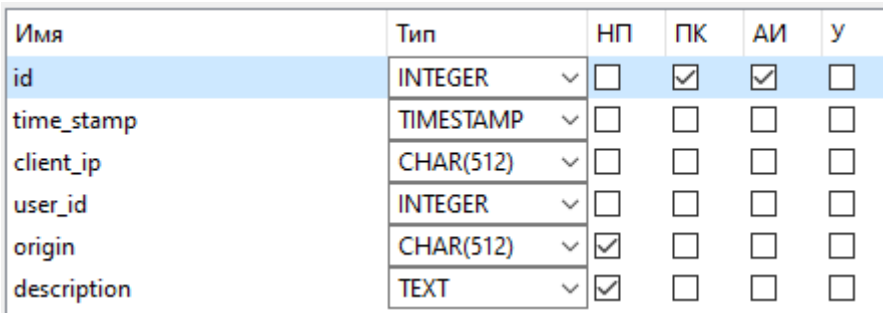
Таблиця “auth_membership” має 2 поля що необхідно заповнити: user_id, group_id (Рисунок 4.5). Поле user_id зберігає ідентифікатор користувача що входить в групу. Поле group_id зберігає ідентифікатор групи в яку входить користувач. Поле id виступає первинним ключем з автоінкрементом. Кожен запис явно ідентифікує окремий зв’язок користувача з групою.



| Имя | Тип | НП | ПК | АИ | У |
|---|---|---|---|---|---|
| id | INTEGER | ☐ | ☑ | ☑ | ☐ |
| role | CHAR(512) | ☑ | ☐ | ☐ | ☐ |
| description | TEXT | ☑ | ☐ | ☐ | ☐ |

Рисунок 4.6 - Параметри таблиці “auth_group”

Таблиця “auth_group” теж має 2 поля які повинні бути заповнені для занесення запису (Рисунок 4.6). Поле role зберігає назву ролі групи. Поле description зберігає текстовий опис ролі для якої призначена створена група користувачів. Поле id - первинний ключем з автоінкрементом.



| Имя | Тип | НП | ПК | АИ | У |
|---|---|---|---|---|---|
| id | INTEGER | ☐ | ☑ | ☑ | ☐ |
| time_stamp | TIMESTAMP | ☐ | ☐ | ☐ | ☐ |
| client_ip | CHAR(512) | ☐ | ☐ | ☐ | ☐ |
| user_id | INTEGER | ☐ | ☐ | ☐ | ☐ |
| origin | CHAR(512) | ☑ | ☐ | ☐ | ☐ |
| description | TEXT | ☑ | ☐ | ☐ | ☐ |

Рисунок 4.7 - Параметри таблиці “auth_event”

Таблиця “auth_event” має два обов'язкових поля  - origin та description (Рисунок 4.7). Поле time_stamp зберігає дані про дату та час типу datetime, коли було задіяно подію входу в систему користувачем. Дані про дату і час зберігаються в одній текствовій змінній типу string. Поле client_ip зберігає IP-адресу з якої було здійснено аутентифікацію користувачем. Поле user_id зберігає ідентифікатор користувача що аутентифікувався. Поле origin зберігає назву об’єкту класу auth. Поле description зберігає опис дії яку виконав користувач – зареєструвався, аутентифікувався або вийшов з системи. Заповнюється таке поле ключовими фразами що створені заздалегідь. Поле id - первинний ключем з автоінкрементом.

| Имя | Тип | НП | ПК | АИ | У |
|---|---|---|---|---|---|
| id | INTEGER | ☐ | ☑ | ☑ | ☐ |
| user_id | INTEGER | ☐ | ☐ | ☐ | ☐ |
| created_on | TIMESTAMP | ☐ | ☐ | ☐ | ☐ |

Рисунок 4.8 - Параметри таблиці “auth_cas”

Таблиця “auth_cas” складається з трьох полів (Рисунок 4.8). Поле id - це стандартне поле для ідентифікації запису з автоінкрементом. Поле user_id зберігає ідентифікатор користувача системи. Поле created_on зберігає дату та час занесення запису до таблиці. Тип даних поля – datetime.

| Имя | Тип | НП | ПК | АИ | У |
|---|---|---|---|---|---|
| id | INTEGER | ☐ | ☑ | ☑ | ☐ |
| id_user | INTEGER | ☑ | ☐ | ☐ | ☐ |
| num_session | CHAR(512) | ☑ | ☐ | ☐ | ☐ |
| id_authEvent | INTEGER | ☑ | ☐ | ☐ | ☐ |
| id_expression | INTEGER | ☑ | ☐ | ☐ | ☐ |
| name_table_expression | CHAR(512) | ☑ | ☐ | ☐ | ☐ |
| answer | CHAR(512) | ☑ | ☐ | ☐ | ☐ |
| is_correct | CHAR(512) | ☐ | ☐ | ☐ | ☐ |
| time | CHAR(512) | ☑ | ☐ | ☐ | ☐ |
| date | CHAR(512) | ☑ | ☐ | ☐ | ☐ |

Рисунок 4.9 - Параметри таблиці “userAnswer”

Таблиця “userAnswer” має 8 обов’язкових полів для заповнення запису: id_user, num_session, id_authEvent, id_expression, name_table_expression, answer, is_correct, time та date (Рисунок 4.9). Поле id_user зберігає ідентифікатор користувача що дав відповідь на завдання тесту. Поле num_session зберігає ідентифікатор сесії, тобто, скільки разів користувач заходив у систему між подіями аутентифікації та виходу з аккаунту системи. Поле id_authEvent зберігає ідентифікатор події аутентифікації користувача. Поле id_expression зберікає ідентифікатор одного завдання з тесту. Поле name_table_expression зберігає назву тесту до якого відноситься завдання на яке дав відповідь користувач. Поле answer зберігає відповідь на завдання тесту. Поле is_correct зберігає інформацію про те, чи правильно дав відповідь користувач на завдання тесту. Значення відповіді користувача з поля answer таблиці “userAnswer”

порівнюється зі значенням поля answer таблиці “tasks”, і якщо відповіді співпадають, то заноситься строка “True” до поля  is_correct таблиці “userAnswer”. Якщо відповідь не співпадає з записом поля answer таблиці “tasks”, то заноситься строка “False”.

Поле таблиці “userAnswer”  time зберігає час, коли користувач дав відповідь на запитання тесту. Поле date зберігає дату, коли користувач дав відповідь на запитання тесту. Поле id - первинний ключем з автоінкрементом.

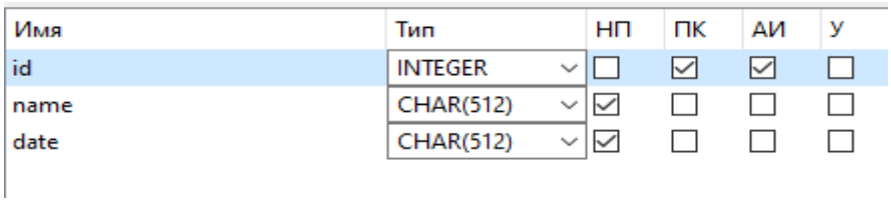



Рисунок 4.10 - Параметри таблиці “tests”

Таблиця “tests” має два поля, що повинні бути заповнені – name та date (Рисунок 4.10). Поле name зберігає назву оголошеного тесту. Поле date зберігає дату дедлайну для тесту. Поле id зберігає ідентифікатор оглошеного тесту, заповнюється автоматично та виступає первинним ключем.

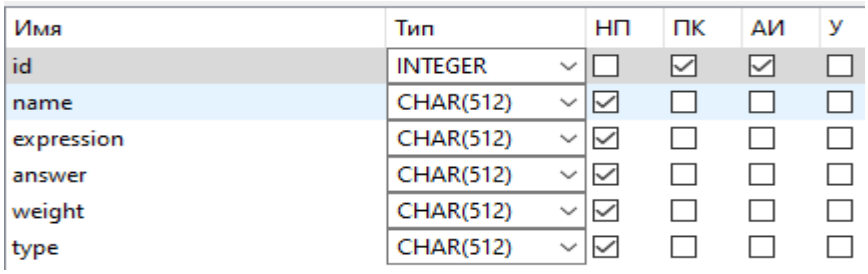



Рисунок 4.11 - Параметри таблиці “tasks”

Таблиця “tasks” має 3 поля що необхідно заповнити перед занесенням запису у таблицю: name, expression, answer (Рисунок 4.11). Поле name зберігає назву тесту до якого відноситься завдання. Поле expression зберігає вираз завдання тесту, на яке користувач повинен дати відповідь. Поле answer зберігає правильну відповідь на завдання тесту. Поле id зберігає кількість балів за завдання, type зберігає тип завдання. Поле id - первинний ключем з автоінкрементом.

# 5 АЛГОРИТМ РОБОТИ СИСТЕМИ АВТОМАТИЗАЦІЇ ТА ОСНОВНІ ФУНКЦІЇ ІНТЕРФЕЙСУ КОРИСТУВАЧА

У даному розділі описується програмні рішення для виконання завдання, пояснюється алгоритм роботи програми для кожного з модулів Model, View, Control. Програмний код системи складається з логіки роботи системи та з графічного інтерфейсу користувача – адміністратора або клієнта. У програмному коді системи не можна виділити основний клас, оскільки він складається з автономних модулів, котрі з деякими змінами у логіці можна використовувати як окремі одиниці.

## 5.1 Модуль Model

У модулі Model будується структура бази даних, оголошуються поля таблиць та визначаються їх параметри, виставляються атрибути.

Для реалізації системи аутентифікації користувачів було обрано клас Auth, який постачається разом з фреймворком web2py. Він дозволяє впроваджувати різноманітні системи розподілення користувачів на групи та дозволяє індивідуалізувати клас шляхом розширення коду під власні потреби без великих затрат часу та зусиль. Разом зі створенням об’єкту класу Auth до бази даних додається набір таблиць що забезпечують основою системи аутентифікації.

Для збереження тестів було створено дві таблиці – “tests” та “auth”. Той факт, що усі завдання кожного тесту об’єднані одією таблицею суттєво спрощує роботу з базою даних, дозволяє зручно фільтрувати та сортувати дані, забезпечує цілісність бази даних під час роботи з програмою.

Історія відповідей на завдання тестів зберігається у створеній таблиці userAnswer.

Створення таблиць відбувається за допомогою методів класу DAL, що постачаються фреймворком. Це дозволяє уникнути використання для створення бази даних стороннього програмного забезпечення.

Будь-які зміни що вносяться до об’єктів таблиць класу DAL поза модулем Model не мають глобального впливу на базу даних, і зберігаються лише в межах блоку коду – цілого файлу, користувацького методу або ж оператору умовного вибору або циклу, під час зчитування коду інтерпретатором Python.

## 5.2 Алгоритм програми для реалізації інтерфейсу користувача

Алгоритм інтерфейсу користувача побудований на взаємодії між модулями Controller і View.

Система авторизації впроваджується до системи декоратором об’єкту класу Auth що передує методу головної веб-сторінки /index з файлу модулю Controller – default.py (Рисунок 5.1).

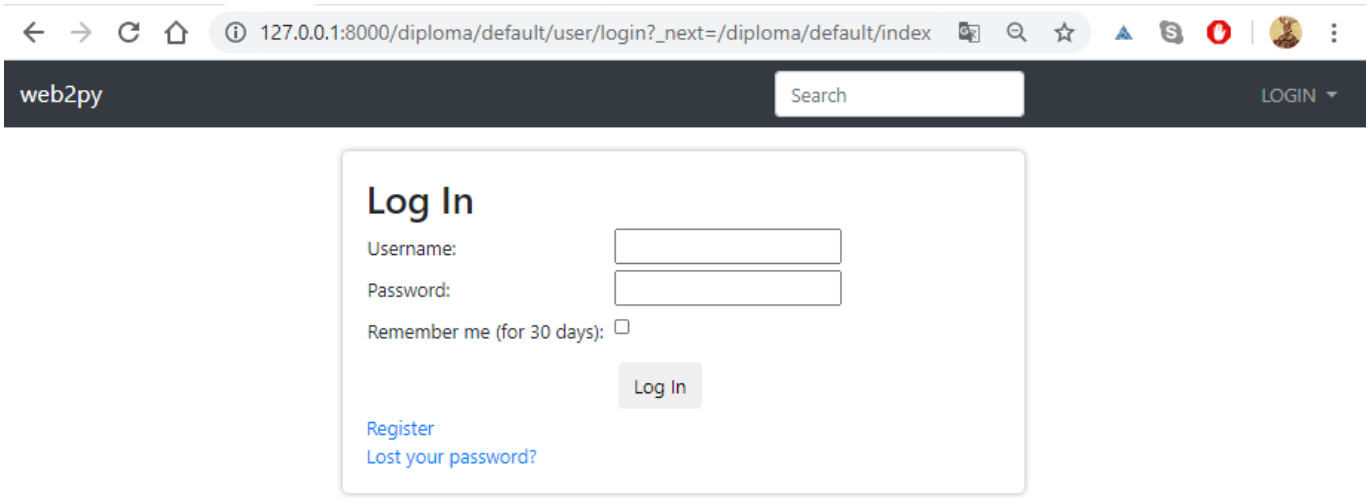



Рисунок 5.1 - Сторінка авторизації користувача

Метод аутентифікації login() повертає на відповідну веб-сторінку форму, яку користувач повинен заповнити. Після чого відбувається порівняння введених користувачем даних з даними у таблиці auth_user на наявність запису про такого зареєстрованого користувача. Якщо порівняння успішне, то користувача система

перенаправляє на головну веб-сторінку  /index. Якщо відповідного запису у таблиці auth_user не знаходиться то система повідомляє користувача про те, що він ввів невірний логін або невірний пароль.

Алгоритм системи реєстрації виконано подібним чином: метод register() так само повертає на сторінку представлення (View) форму, яку повинен заповнити користувач, після чого зчитує введені дані, і якщо вони відповідають вимогам, заносить запис про нового зареєстрованого користувача до таблиці auth_user, де знаходитимуться його логін, пароль, електронна пошта та прізвище з ім’ям (Рисунок 5.2). До таблиці auth_event робиться запис про те, що користувач зареєструвався у системі. Запис зберігає дату, час реєстрації і IP-адресу користувача.

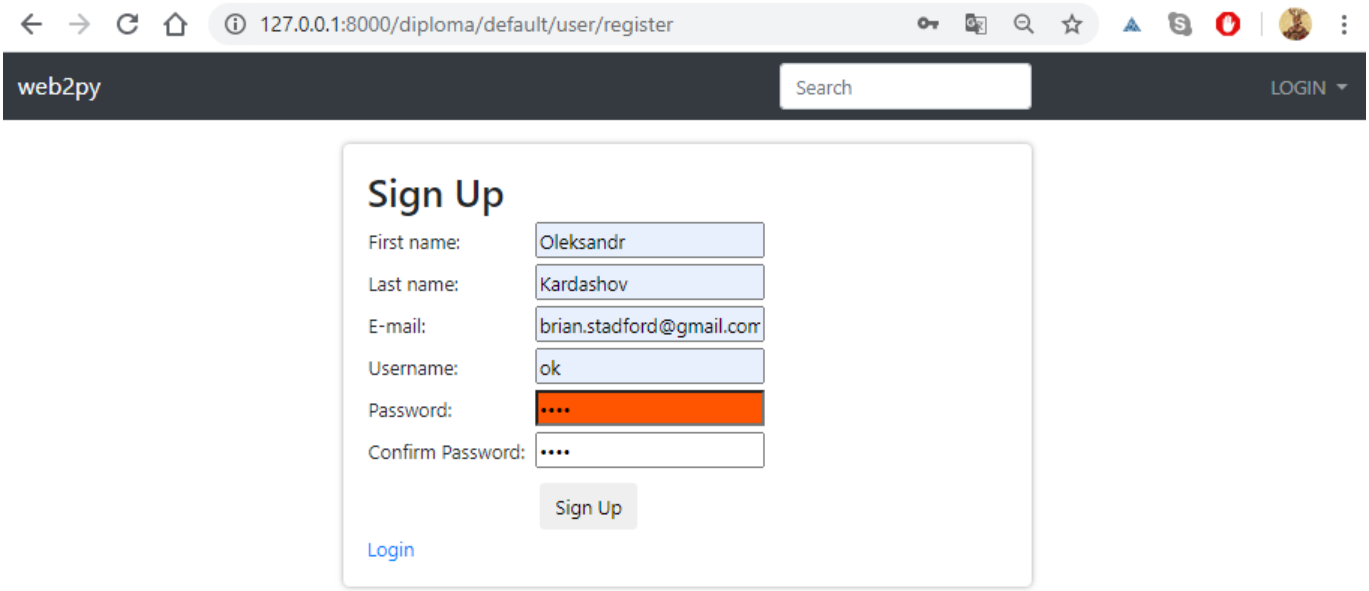



Рисунок 5.2 - Сторінка авторизації користувача

Оскільки у поля E-mail, Username повинні бути унікальними, то якщо система знаходить збіг по ним під час реєстрації нового користувача, то виведе на веб-сторінку відповідне повідомлення про те, що така електронна адреса або логін вже використовується (Рисунок 5.3).

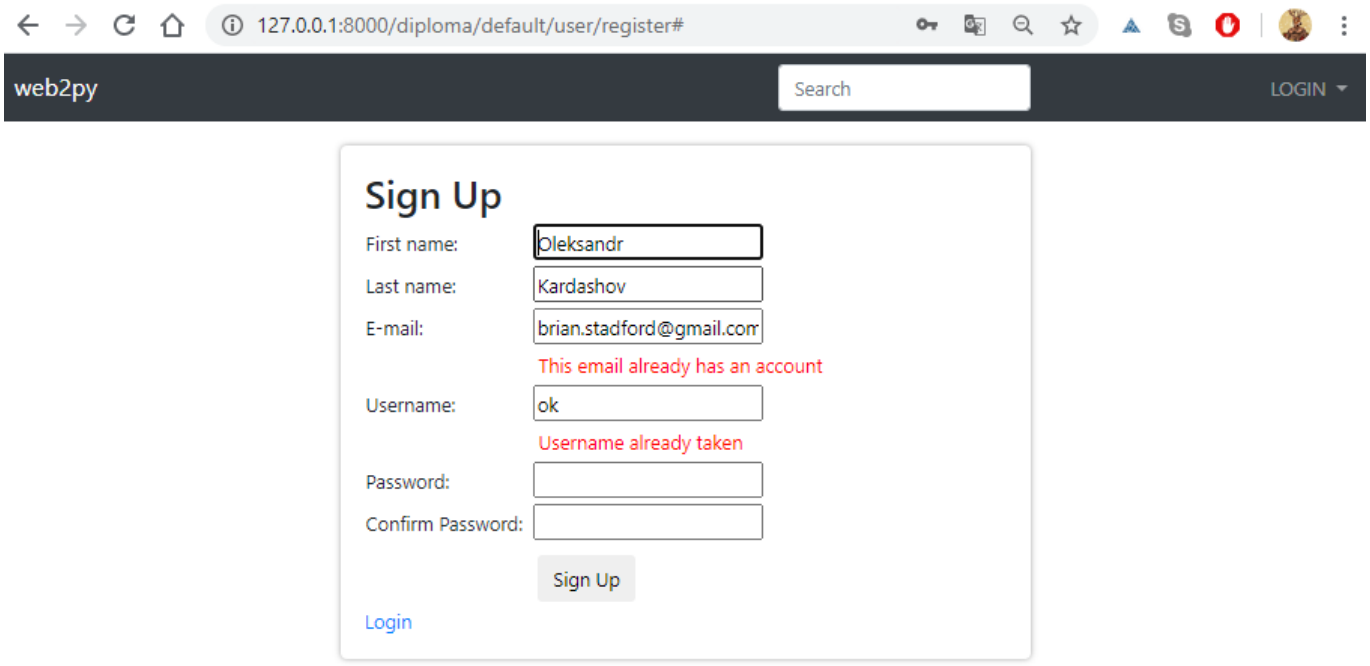



Рисунок 5.3 – Демонстрація повідомлень для відповідних виключень

Розмітка головної сторінки відбувається повністю у файлі представлення default.html. Відбувається циклічне зчитування списку створених тестів з таблиці “tests” і на інтерфейсі відображається відповідна кнопка з посиланням на сторінку проходження обраного тесту. Завдяки тому що список створених тестів винесено в окрему таблицю, такі алгоритми як вивід даних про тести зводяться до одного циклу, що значно прискорює процес завантаження сторінки, оскільки не виконуються зайві перевірки на умову повторення назв тестів і не відбувається зайвих циклічних проходжень.

Виведення переліку тестів динамічно реагує на зміну розміру екрану. В одному рядку виводиться по три посилання. На Рисунку 5.4 наведено чотири тести які студент може пройти:

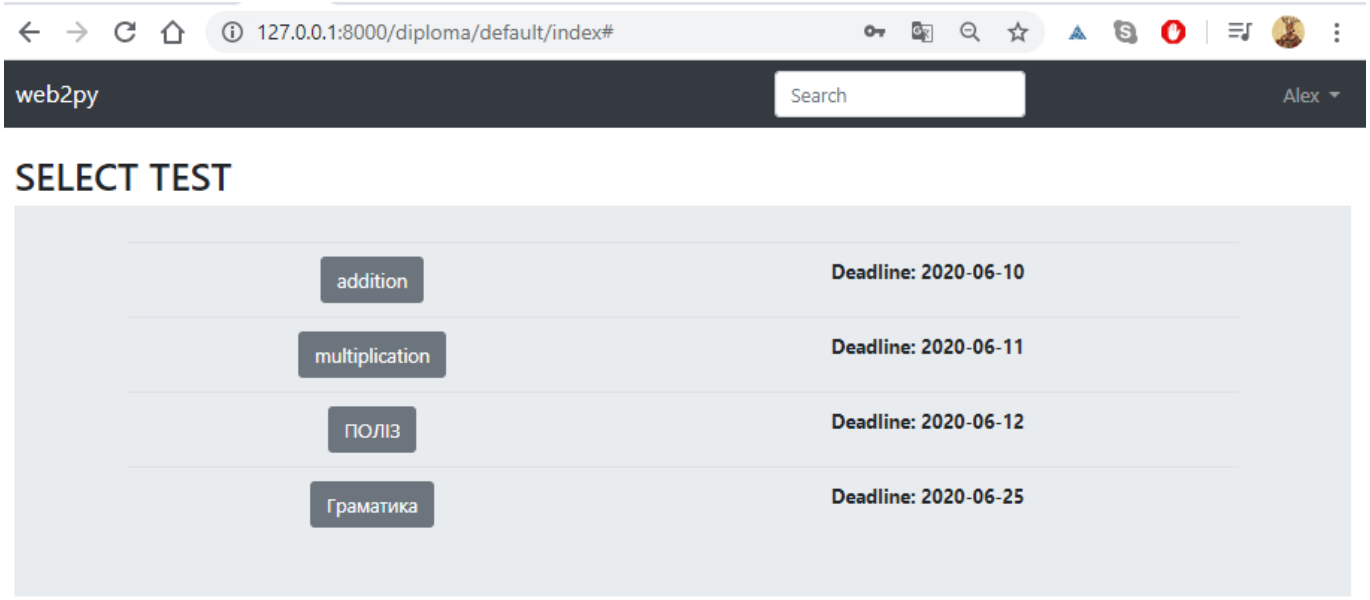



Рисунок 5.4 - Сторінка вибору тестів користувачем

Згідно записів у таблиці бази даних “tests” відбувається пошук серед переліку тестів і знайдений тест потрапляє до форми, що пересилається до файлу представлення default.html, котрий зчитуючи її виводить на інтерфейс відповідну кількість елементів керування – кнопок, які відповідають за кожен доступний тест.

При натисканні  на одне з посилань відбувається перенаправлення користувача на веб-сторінку /result, і разом з тим формується пост-запит, що містить змінну id – ідентифікатором для конкретного тесту.

Використовуючи передані в пост-запиті змінні та аргументи користувацький метод select() створює форму з завданнями обраного тесту. Формування кожного рядка відбувається шляхом заповнення масиву з набором даних (Рисунок 5.5):

- порядковий номер рядка з завданням;
- вираз відповідного завдання;
- текстова строка “Answer”;
- текстове поле введення відповіді користувача;

Кількість таких елементів масиву відповідає кількості завдань у таблиці “tasks”, що знайдені за ключовою назвою обраного тесту.

Після формування масиву даних він передається до конструктору об’єкта класу FORM(). Створена форма передається до файлу представлення default.html, після чого

очікує на натискання користувачем на кнопку “Отправить”. При натисканні на кнопку відбувається зчитування даних що ввів користувач і викликається метод класу DAL insert() який застосовується для таблиці userAnswer, де треба внести значення усіх полів таблиці до якої необхідно зробити запис.

Разом з відповідями до таблиці заноситься інформація про дату та час проходження тесту студентом та відбувається перевірка на правильність даних користувачем відповідей. З кожного вікна вводу тексту, що має окреме ім’я, зчитується текстова строка і порівнюється з правильною відповіддю на завдання, що зберігається у таблиці “tasks”. Пошук потрібної правильної відповіді у таблиці “tasks” відбувається за аргументом-ім’ям тесту, що був надісланий у пост-запиті з попередньої сторінки представлення /index. Якщо текстові строки з поля вводу та зі знайденого запису з таблиці “tasks” співпадають, то до методу insert(), що формує новий запис про відповідь користувача на завдання надходить змінна зі значенням True. Якщо строки не співпадають, то до методу insert() надходить змінна зі значенням False.

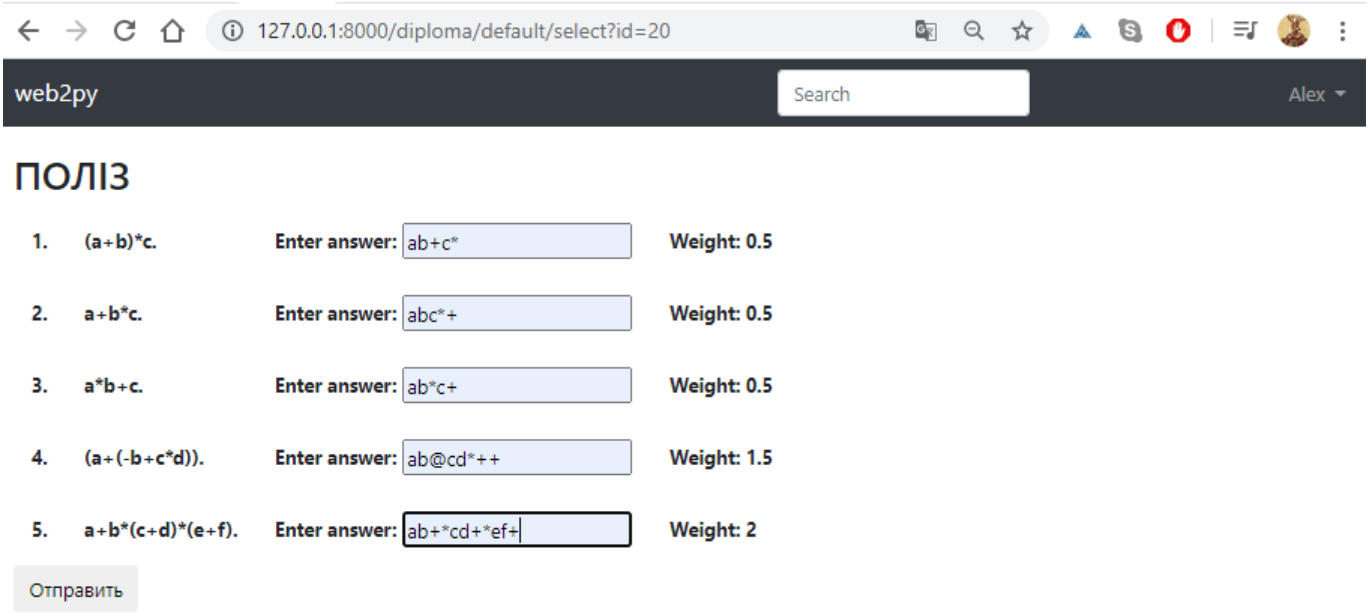



Рисунок 5.5 - Сторінка проходження обраного тесту користувачем

При перенаправленні користувача на сторінку виведення результатів тестування формується пост-запит з аргументом, що зберігає текстову строку з

ідентифікатором імені тесту, завдяки чому можна виводити його на веб-сторінку і виконувати пошук занесених до таблиці з відповідями користувачів “userAnswer” даних. Пошук та ідентифікація щойно зроблених користувачем відповідей відбувається за такими полями таблиці “userAnswer”, як id_authEvent та num_session. Комбінація ідентифікаторів подій аутентифікації конкретного користувача та номеру сесії явно ідентифікують щойно зроблені записи з відповідями серед інших. Робиться вибірка рядків з таблиці “userAnswer”, за котрими відбувається циклічне проходження і на кожній ітерації виводиться сформований рядок що містить наступні параметри (Рисунок 5.6):

- порядковий номер відповіді відповідно до завдання тесту;
- завдання тесту;
- відповідь користувача;
- інформація про правильність зробленої відповіді;
- яку кількість балів було отримано за завдання;

Якщо тест було виконано невчасно, то загальна отримана кількість балів множиться на коефіцієнт 0.6.

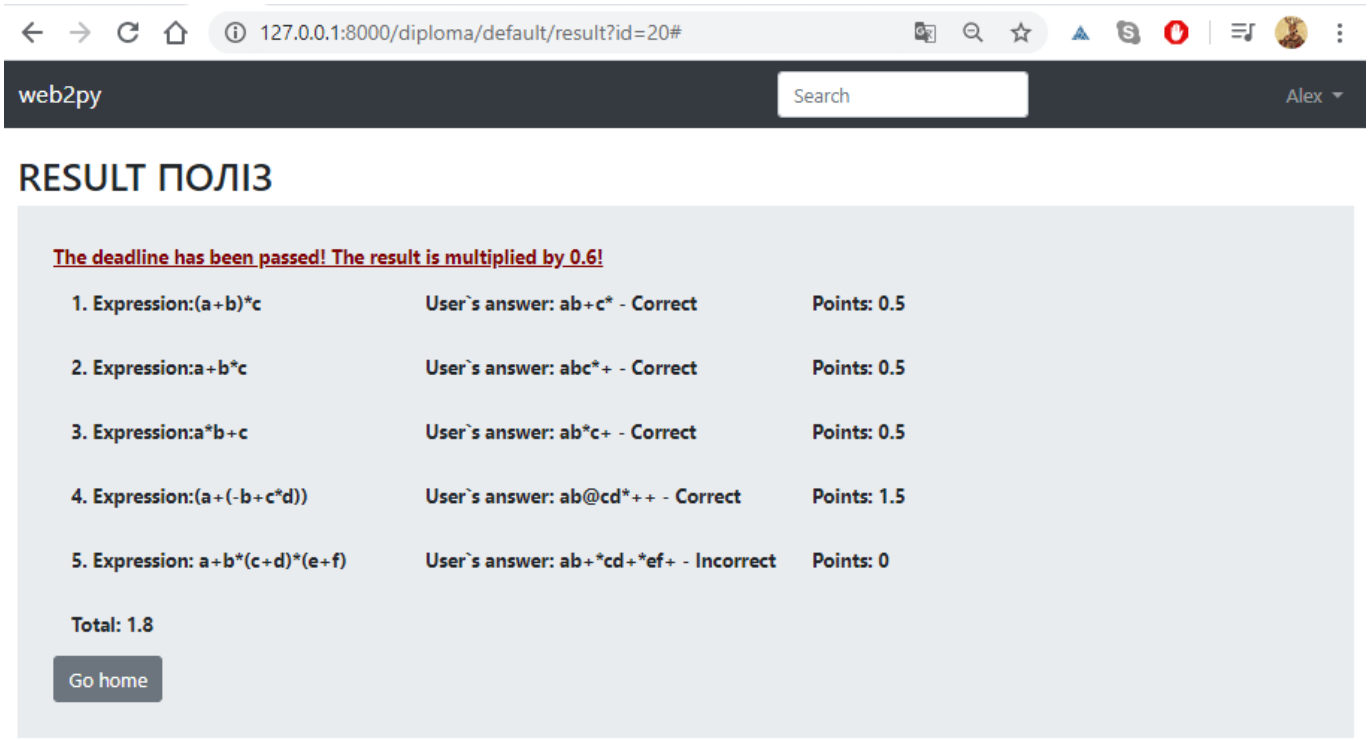



Рисунок 5.6 - Сторінка виведення результатів тестування

Якщо обраний студентом тест є завданням на написання граматики у нотації Бекуса-Наура, то у такому випадку сторінка виконання завдання виглядатиме як поле, де студент повинен буде написати граматику, наприклад, для мови програмування, що містить різні арифметичні оператори, підтримує декілька типів змінних, що зображено на Рисунку 5.7.

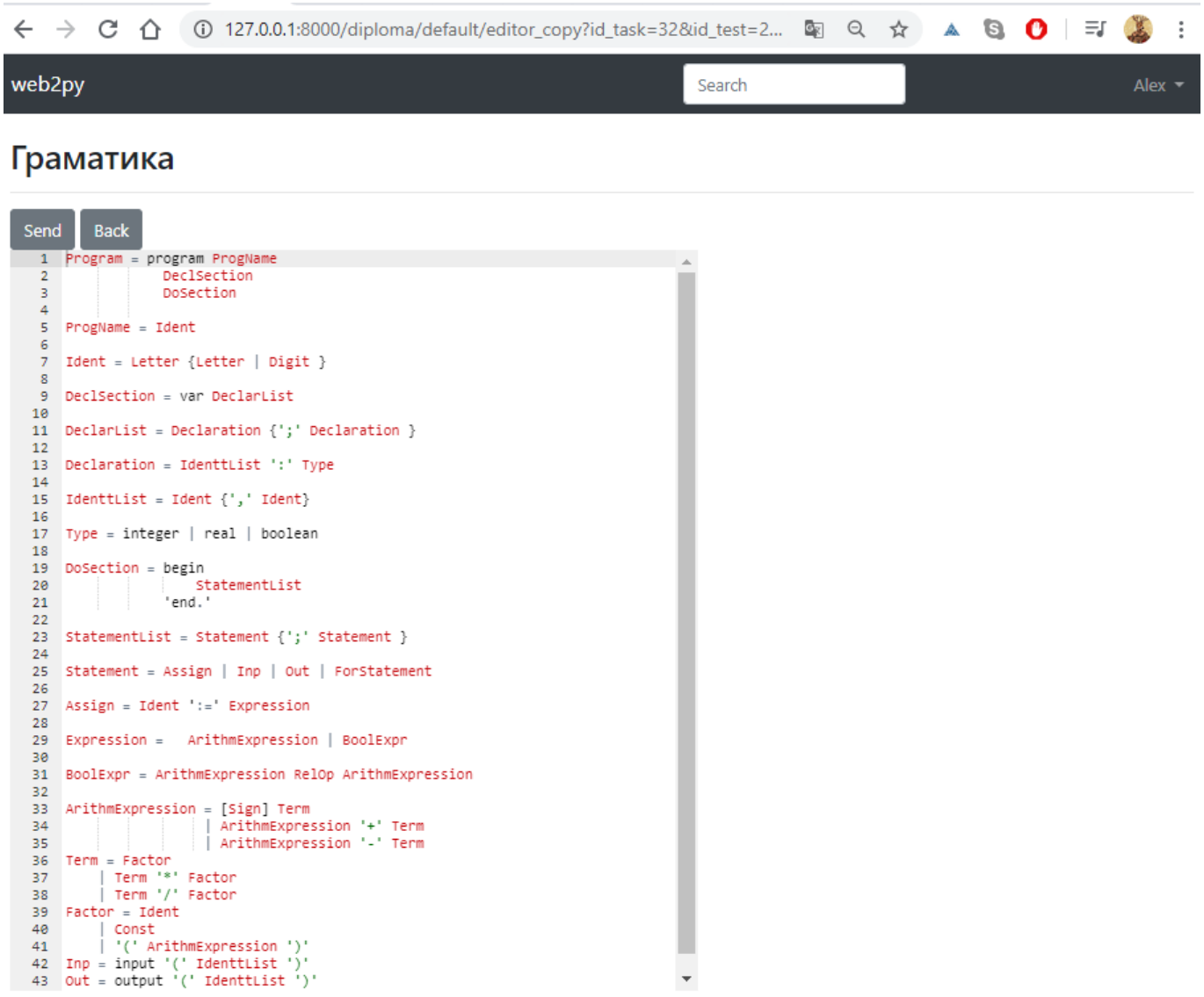



Рисунок 5.7 – Сторінка завдання з написання граматики

Текст, що вводить користувач обробляється у реальному часі та підсвічується червоним кольором, якщо слово є нетерміналом, і залишається чорним, якщо слово є терміналом. Для обробки введених даних було використано редактор коду Ace, що є вільним у використанні.

Кнопка-посилання внизу веб-сторінки /result дозволяє користувачу повернутися на головну сторінку системи для подальшого вибору та проходження тестів. Після повернення користувача на головну сторінку лічільник сесій інкрементується і при другій спробі проходження тесту так само можна буде явно ідентифікувати зроблені користувачем відповіді для виведення на веб-сторінку результатів тестування.

У будь-який момент роботи з системою, користувач має змогу вийти зі свого аккаунту на сторінку аутентифікації користувача, при цьому до таблиці auth_event занесеться запис про вихід користувача з конкретним ідентифікатором з системи.

Користувач має змогу відредагувати дані свого аккаунту якщо перейде за відповідним посиланням у випадаючому вікні у заголовку будь-якої сторінки (Рисунок 5.8). Система перенаправляє користувача на сторінку /profile, де будується форма котра складається з доступних для модифікації полів імені, прізвища та логіну користувача. Електронну пошту змінити не можна. При натисканні на кнопку-посилання “Appy changes”, згенерується команда на оновлення запису з таблиці auth_user за допомогою методу класу DAL - update(), де аргументи це необхідні для редагування поля запису.

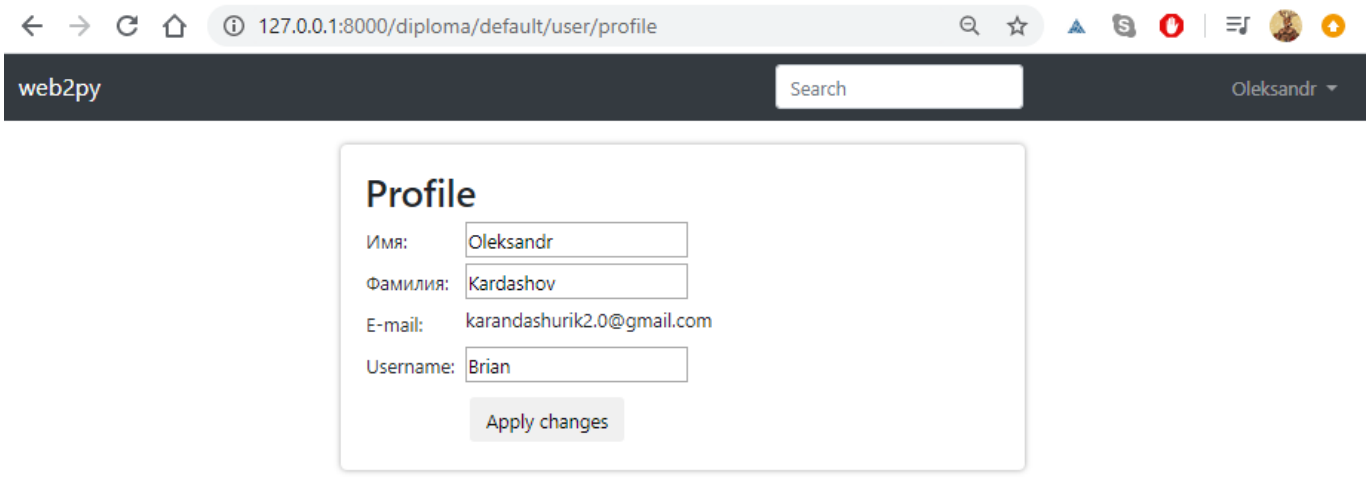



Рисунок 5.8 – Сторінка редагування профілю користувача

## 5.3 Алгоритм програми для реалізації інтерфейсу адміністратора

Панель адміністратора створено для надання можливості адміністратору системи створювати, редагувати та видаляти тести, та завдання кожного з тестів. Також адміністратор може переглядати історію відповідей користувачів, фільтрувати надані системою дані та сортувати їх.

Розмітка головної сторінки повністю будується у файлі представлення admin.html. Містить два посилання-кнопки, одне на сторінку редагування тестів, друге на сторінку перегляду історії відповідей користувачів на завдання тестів (Рисунок 5.9).

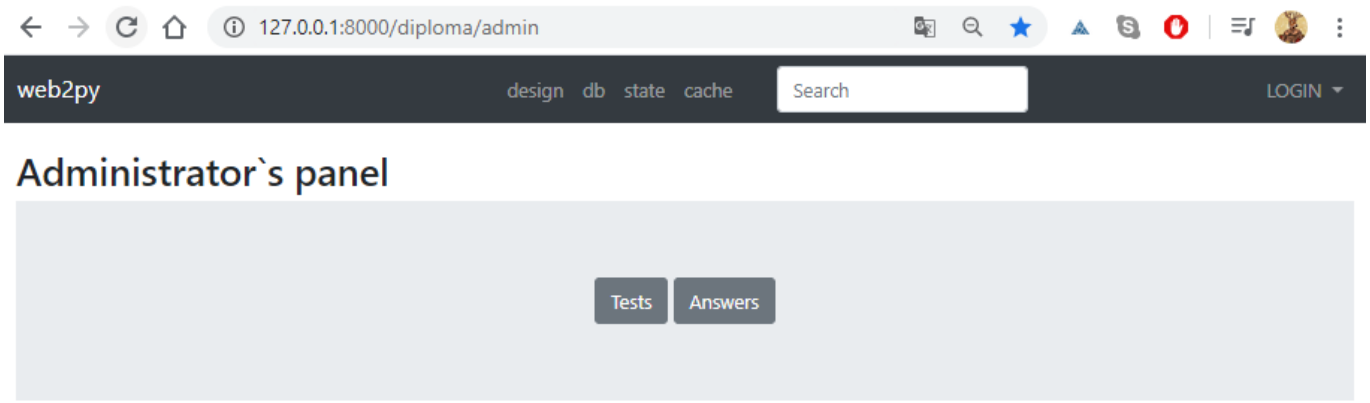



Рисунок 5.9 – Головна сторінка панелі адміністратора

### 5.3.1 Алгоритм сторінки з перегляду переліку тестів системи

При переході адміністратора на веб-сторінку редагування тестів /showTests у файлі представлення admin.html виводиться список усіх створених тестів. Для формування вибірки викликається метод select() класу DAL для таблиці “tests”, і вибірка даних присвоюється до масиву даних. Побудова розмітки сторінки відбувається циклічно проходячи по масиву з даними вибірки. Для кожного рядка вибірки на веб-сторінку виводиться (Рисунок 5.10):

- посилання на редагування тесту;
- посилання на  сторінку додавання нового завдання до тесту;
- посилання на редагування дедлайну тесту;
- посилання на сторінку перейменування тесту;

- посилання на сторінку видалення тесту;

Окремо над списком тестів знаходиться два посилання: на додавання нового тесту до таблиці, та на повернення адміністратора на головну сторінку панелі адміністратора.

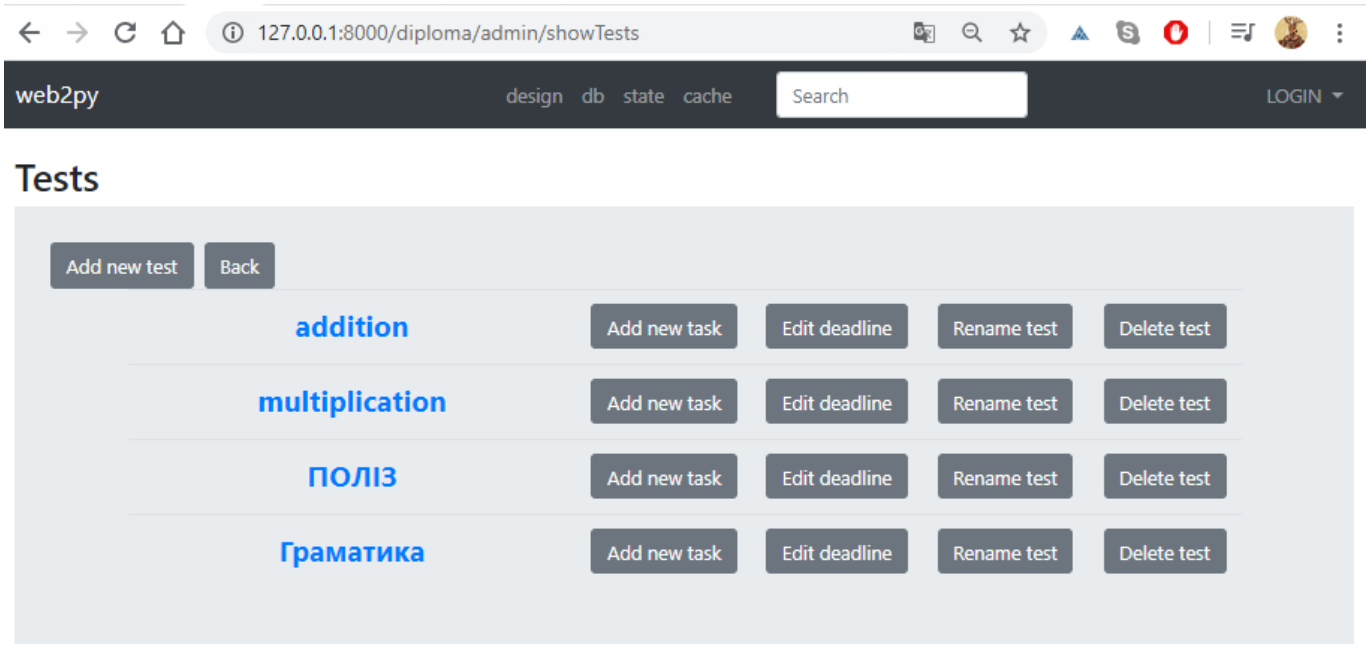



Рисунок 5.10 – Сторінка для перегляду списку тестів системи

При переході адміністратора за посиланням для додавання нового тесту до списку, викликається метод addTest() модуля Controller з файлу admin.py. У ньому формується форма з поля для введення тексту, кнопки-відправки форми та кнопки-посилання на попередню сторінку перегляду списку тестів.

При натисканні адмінісртатором на кнопку відправки форми зчитується введений текст та викликається метод insert(), котрий заносить новий запис до таблиці “tests” бази даних. Після цього системи повертає адміністратора на ту-ж сторінку для створення нового тесту і виводить push-повідомлення про те що тест було створено. Адміністратор може продовжити створювати нові тести (Рисунок 5.11) або може повернутися на попередню сторінку перегляду списку тестів.

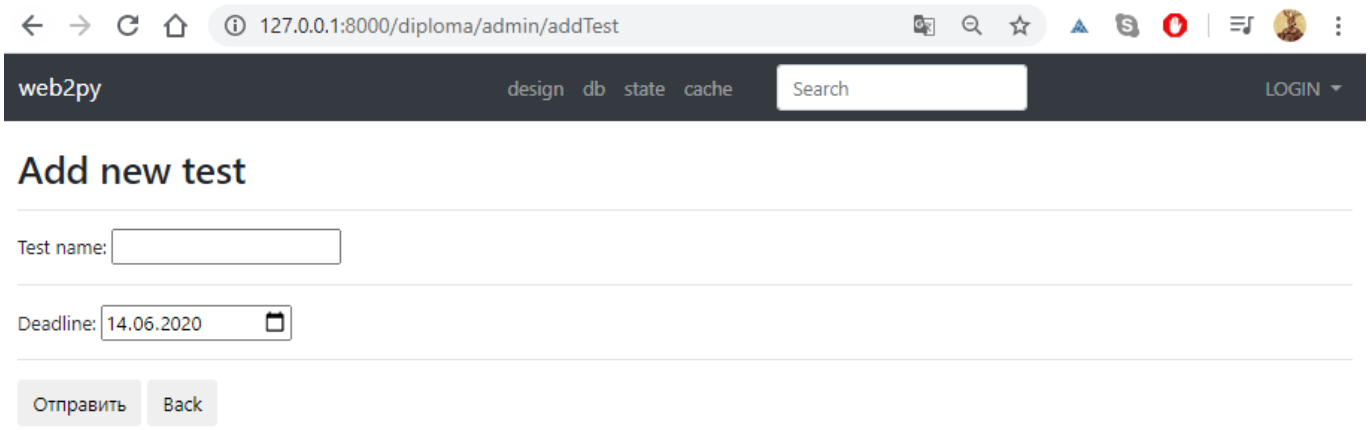



Рисунок 5.11 – Сторінка для створення нового тесту

### 5.3.2 Алгоритм сторінки з перейменування тесту

При натисканні на кнопку посилання на сторінку з редагування назви тесту генерується пост-запит який містить аргумент з назвою тесту та змінну “id”, до якої присвоюється ідентифікатор запису таблиці “tests” з даним тестом (Рисунок 5.12).

Викликається метод editTest() з контролеру admin.py і генерується форма зі схожим змістом до форми додавання нового тесту, з тією різницею, що у текстовому полі виводиться строка-підказка для користувача для нагадування якою була назва обраного ним для перейменування тесту.

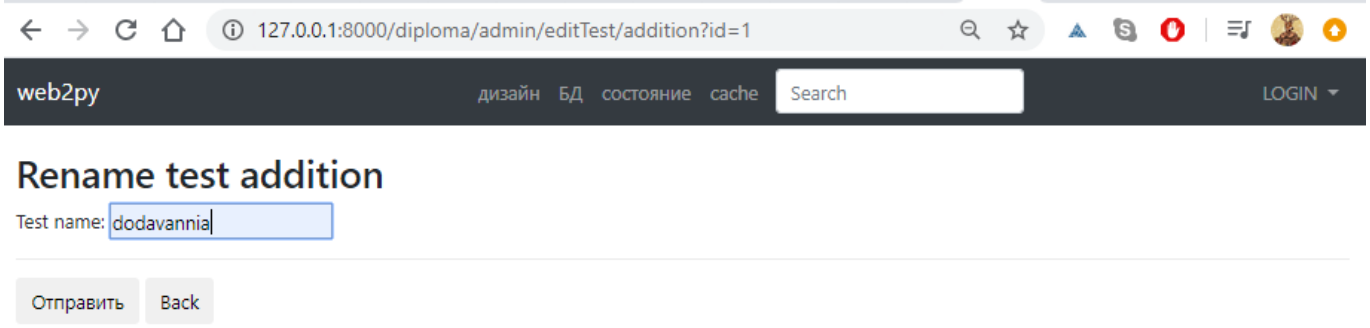



Рисунок 5.12 – Сторінка для редагування назви тесту

Після відправки форми адміністратору надається можливість підтвердити бажання змінити назву тесту, щоб уникнути небажаних незворотних змін у базі даних унаслідок випадкової зміни назви тесту. Потім викликається метод update() який вносить зміну ім’я до запису таблиці “tests”. Так само, метод update() викликається

для таблиці “userAnswer”, де за умови збігу поля з назвою тесту у записі з ім’ям тесту що редагується, вноситься зміна до запису і назва тесту перезаписується. Отже усі записи з історії відповідей користувачів зі старим ім’ям тесту будуть оновлені.

### 5.3.3 Алгоритм сторінки з видалення тесту

Для сторінки видалення тесту немає представлення та розмітки. При проходженні за посиланням на видалення тесту з системи генерується пост-запит що містить аргумент з назвою тесту, наприклад, “delete It ” (Рисунок 5.13), та ідентифікатор поля з таблиці “tests”, id.

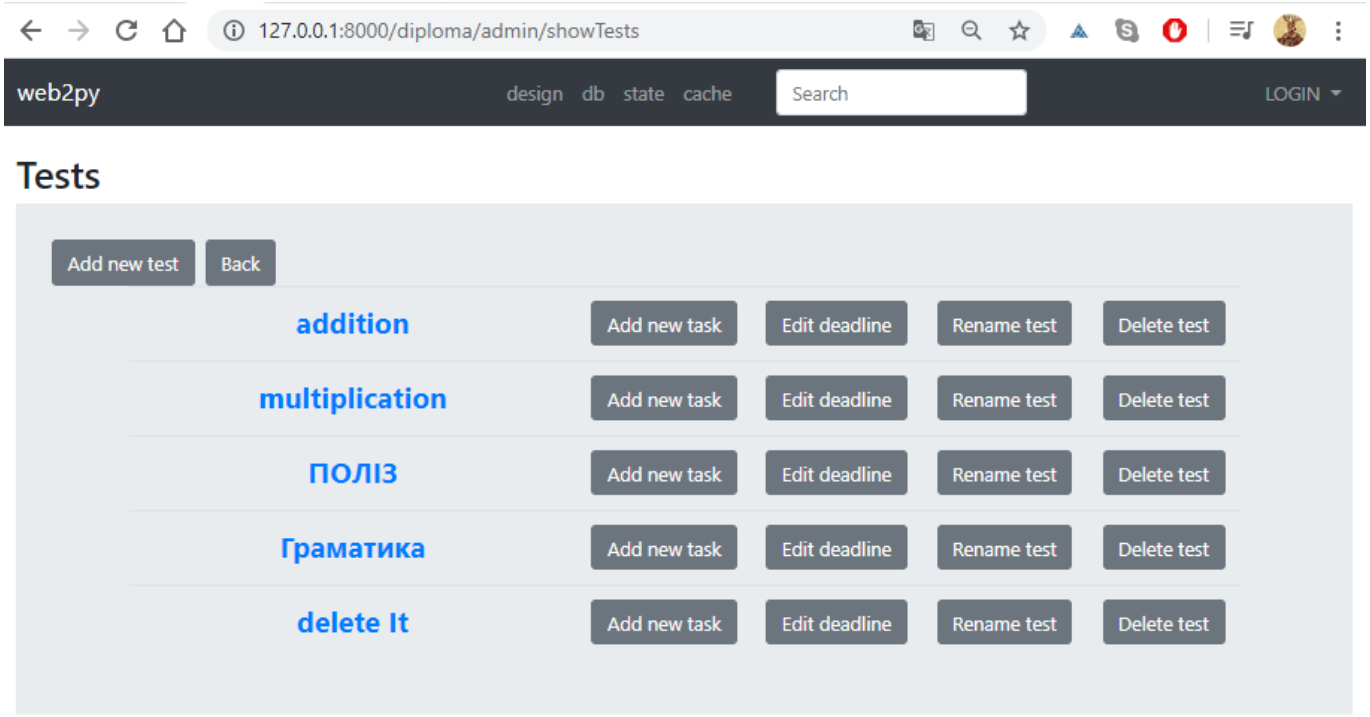



Рисунок 5.13 – Сторінка перегляду тестів до видалення тесту

Системою викликається два рази метод delete() класу DAL, аргументом якого є умова вибору полів таблиці. У даному випадку це ідентифікатор тесту для таблиці “tests” та ім'я тесту для таблиці “userAnswer”. Метод видаляє усі поля двох таблиць де є поле з ідентифікатором id тесту та поле з назвою обраного тесту. Перед видаленням адміністратор повинен підтвердити свою дію видалення тесту з системи для уникнення небажаних незворотних змін до бази даних (Рисунок 5.14).

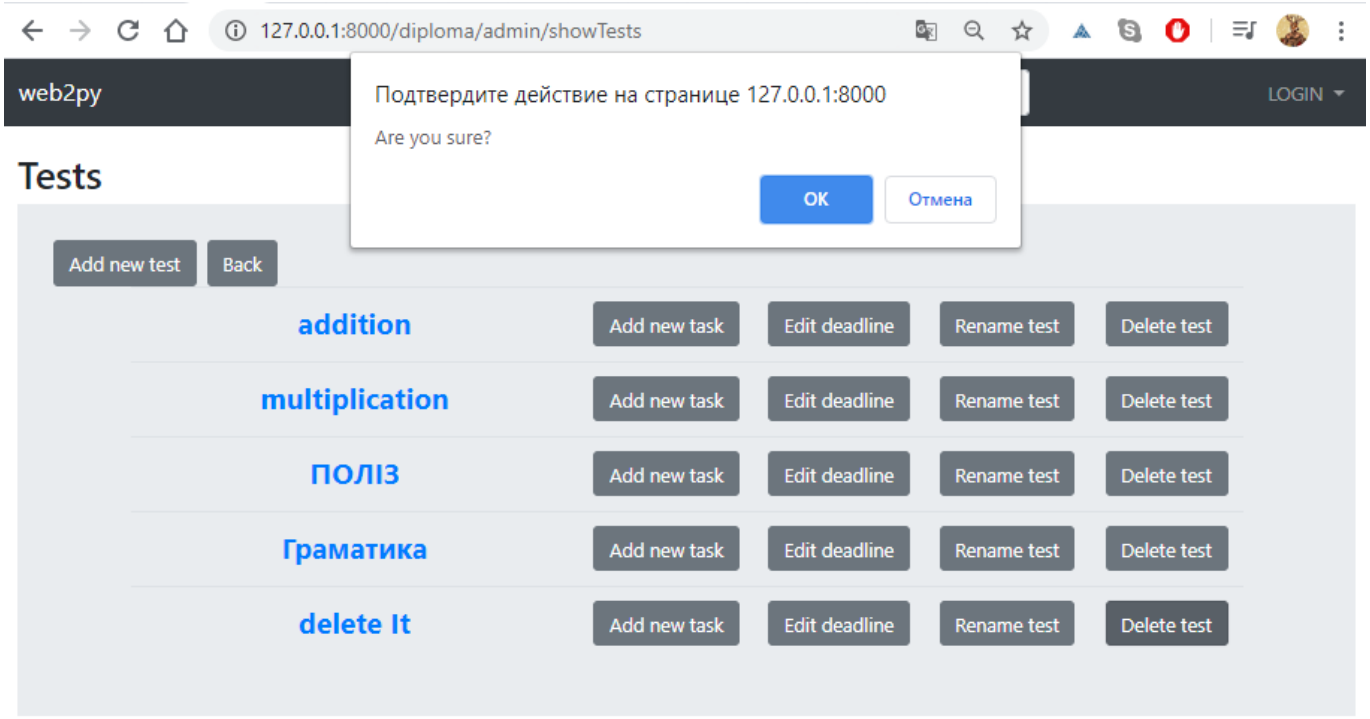



Рисунок 5.14 – Вікно підтвердження дії видалення тесту

### 5.3.4 Алгоритм сторінки перегляду списку завдань тесту

Якщо адміністратор обирає посилання перегляду певного тесту на сторінці перегляду списку тестів, то система перенаправляє його на сторінку перегляду списку завдань обраного тесту /select.

Формується пост-запит що містить в якості аргументу назву обраного тесту та змінну query, до якої присвоюється умова для вибору даних з таблиці “tasks”. Функціонал сторінки перегляду завдань тесту повністю реалізовано у файлі представлення admin.html. Згідно з назвою обраного тесту, робиться запит до таблиці бази даних “tasks” , викликається метод select() класу DAL і формується вибірка даних що циклічно виводить по рядкам наступну інформацію (Рисунок 5.15):

- лічільник рядків;
- завдання тесту;
- правильна відповідь на завдання тесту;
- вага у балах для кожного завдання;
- тип завдання;

- посилання на редагування завдання;
- посилання на видалення завдання;

Над вибіркою даних адміністраторові надається два посилання: на сторінку додавання нового завдання до тесту, та на повернення на попередню сторінку перегляду списку тестів системи.

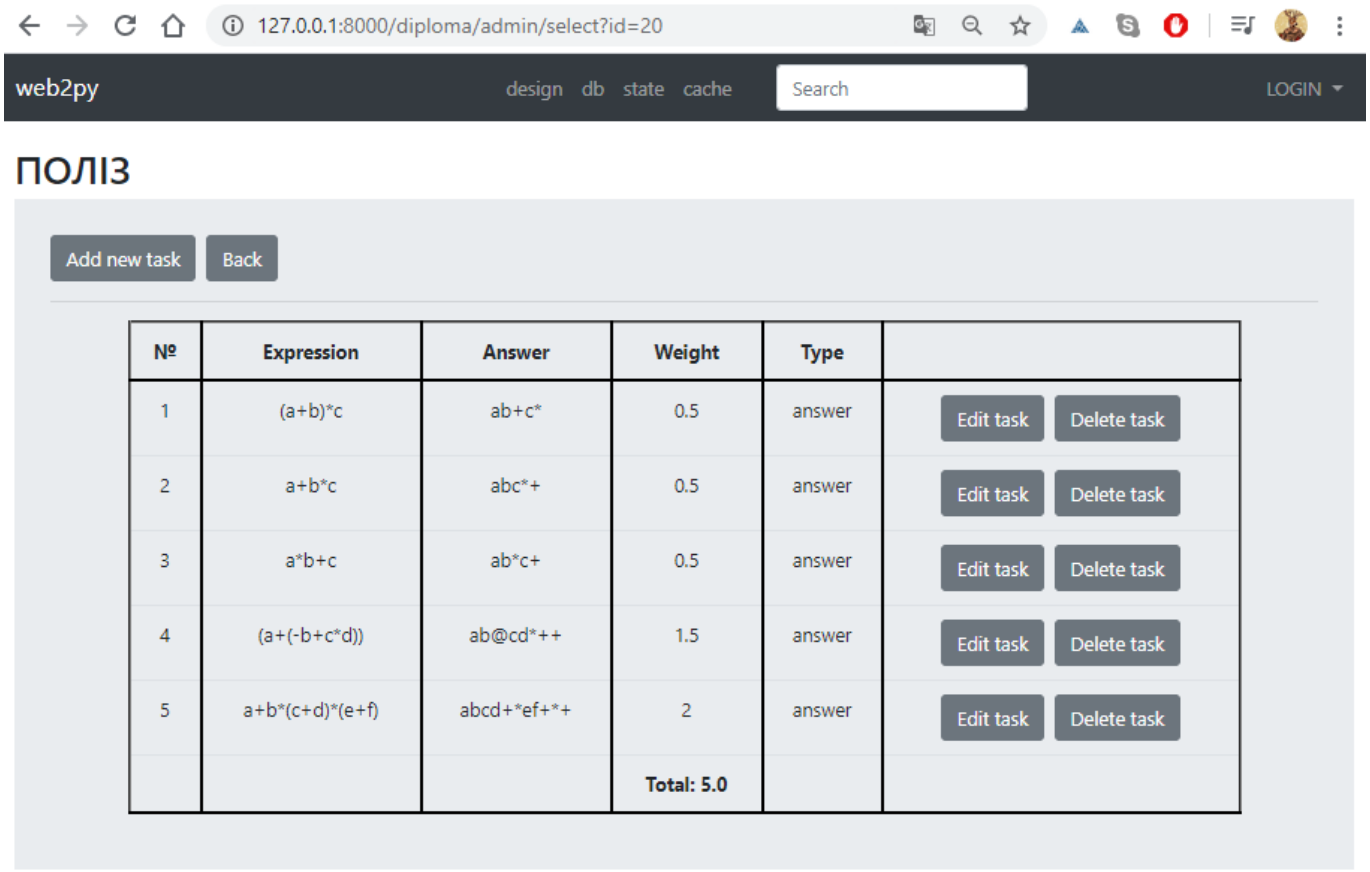



Рисунок 5.15 – Сторінка перегляду завдань обраного тесту

### 5.3.5 Алгоритм сторінки додавання нового завдання тесту

Якщо адміністратор натискає на посилання додавання нового завдання тесту, то генерується пост запит що складається з аргументу, котрий містить назву тесту, після чого адміністратор перенаправляється на сторінку /addTask (Рисунок 5.16).

Система викликає метод addTask() з файлу контролера admin.py, де генерується форма для заповнення адміністратором, що складається з полів введення завдання, та правильної відповіді на завдання. Форма також містить посилання на попередню сторінку перегляду списку завдань обраного тесту на кнопку відправки форми.

Якщо адміністратор відправляє форму, то введена ним інформація зчитується з текстових полів, формується запит до бази даних і викликається метод insert() який заносить до таблиці “tasks” новий запис.

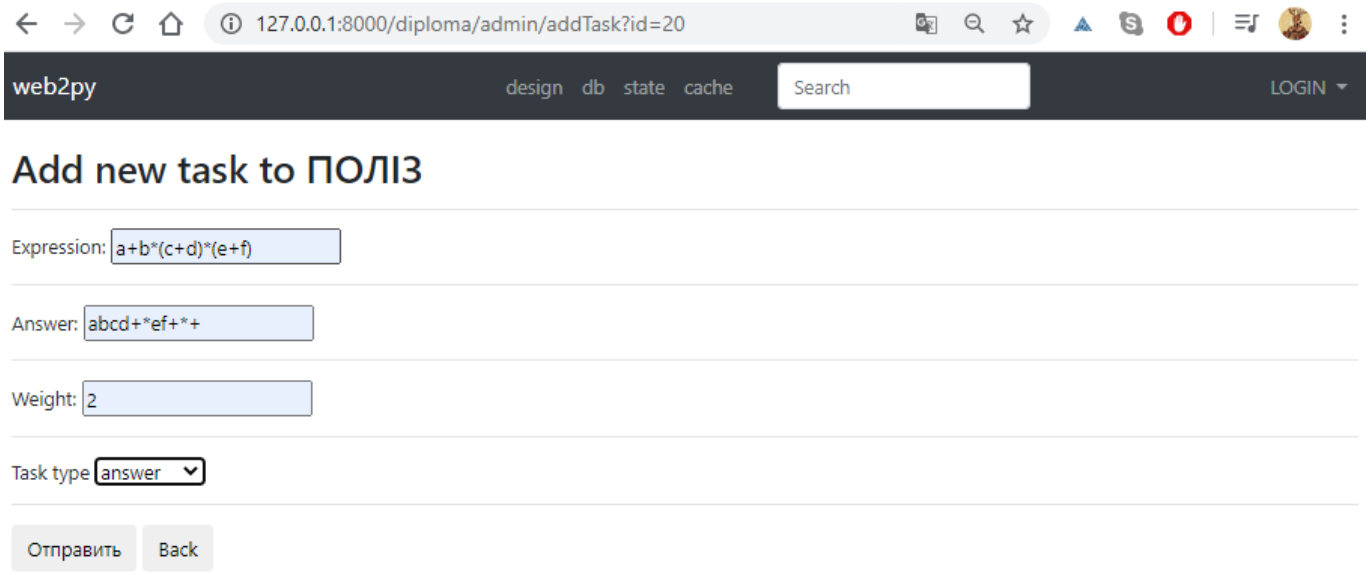



Рисунок 5.16 – Сторінка додавання нового завдання до тесту

Після додавання завдання адміністратор перенаправляється на ту-саму сторінку для додавання нового завдання до тесту для зручного процесу створення тесту.

### 5.3.6 Алгоритм сторінки редагування завдання тесту

Після натискання адміністратором на посилання на редагування обраного завдання тесту, формується пост-запит що складається з аргументу з назвою тесту та змінної id яка містить ідентифікатор обраного завдання, що береться з відповідного запису таблиці “tasks” (Рисунок 5.17).

Адміністраторові на цій сторінці необхідно внести до форми що генерується у методі editTask() нове завдання тесту та правильну відповідь на нього, після чого підтвердити відправку форми.

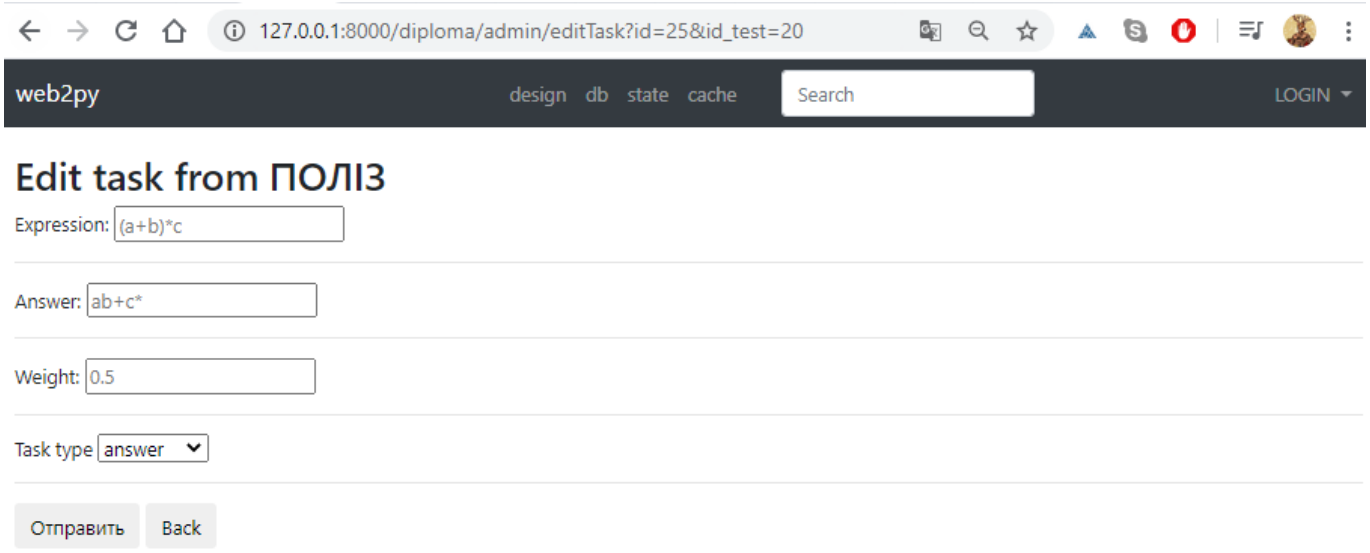



Рисунок 5.17 – Сторінка редагування завдання тесту

### 5.3.7 Алгоритм сторінки видалення завдання тесту

Система формує пост-запит що складається з ідентифікатора id запису таблиці “tasks” після натискання адміністратором на посилання на видалення запису з тесту. Методом deleteTask() контролера admin.py генерується SQL-запит до бази даних на видалення обраного завдання. Для виконання цієї дії адміністратор повинен підтвердити дію на спливаючому вікні. Записи з історії відповідей на завдання не видаляються оскільки присутність у таблиці “userAnswers” видаленого завдання не створює неоднозначності у таблиці.

## 5.4 Алгоритм системи перегляду історії відповідей користувачів

Система надає можливість перегляду історії відповідей на тести. Якщо адміністратор натискає на посилання “Answers” на головній сторінці панелі адміністратора /index, то система перенаправляє його на сторінку /showAnswers.

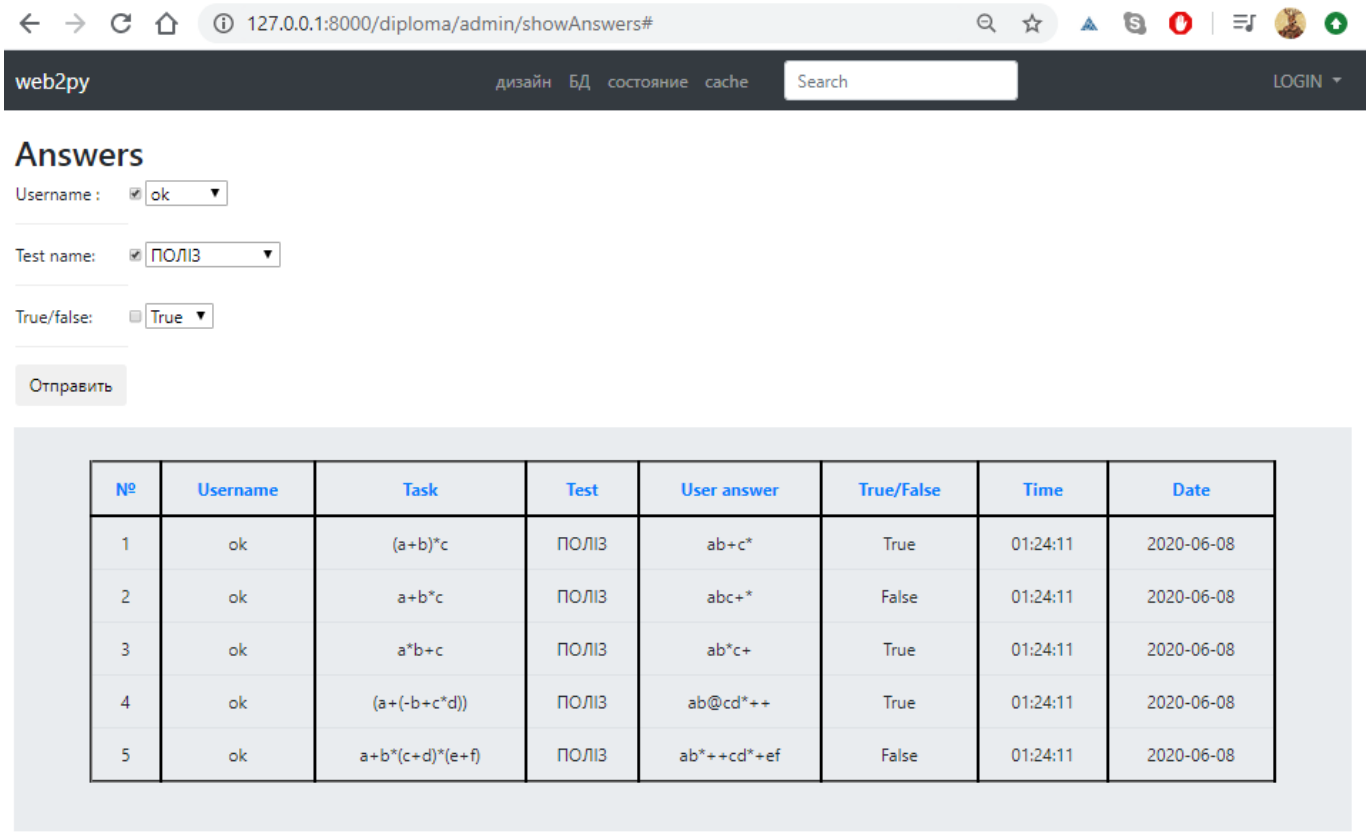
### 5.4.1 Алгоритм фільтрації даних історії відповідей

Викликається метод showAnswers() контролера admin.py, який генерує форму що містить фільтри для даних що виводяться у файлі представлення admin.html. Форма складається з трьох фільтрів (Рисунок 5.18):

- за логіном користувача;
- за назвою тесту;
- за правильністю відповіді користувача;

Кожен фільтр містить checkbox – поле що відповідає за активність даного фільтру та список параметрів які можна застосувати до даного фільтру. Для логіну - це список усіх зареєстрованих користувачів. Для назви тесту - це список усіх створених тестів. Для правильності відповіді - це параметри True або False.

Якщо активовано один або більше фільтрів, достатньою відправити форму щоб фільтри застосувалися до вибірки даних відповідно до обраних параметрів.




| № | Username | Task | Test | User answer | True/False | Time | Date |
|---|---|---|---|---|---|---|---|
| 1 | ok | (a+b)*c | ПОЛІЗ | ab+c* | True | 01:24:11 | 2020-06-08 |
| 2 | ok | a+b*c | ПОЛІЗ | abc+* | False | 01:24:11 | 2020-06-08 |
| 3 | ok | a*b+c | ПОЛІЗ | ab*c+ | True | 01:24:11 | 2020-06-08 |
| 4 | ok | (a+(-b+c*d)) | ПОЛІЗ | ab@cd*++ | True | 01:24:11 | 2020-06-08 |
| 5 | ok | a+b*(c+d)*(e+f) | ПОЛІЗ | ab*++cd*+ef | False | 01:24:11 | 2020-06-08 |

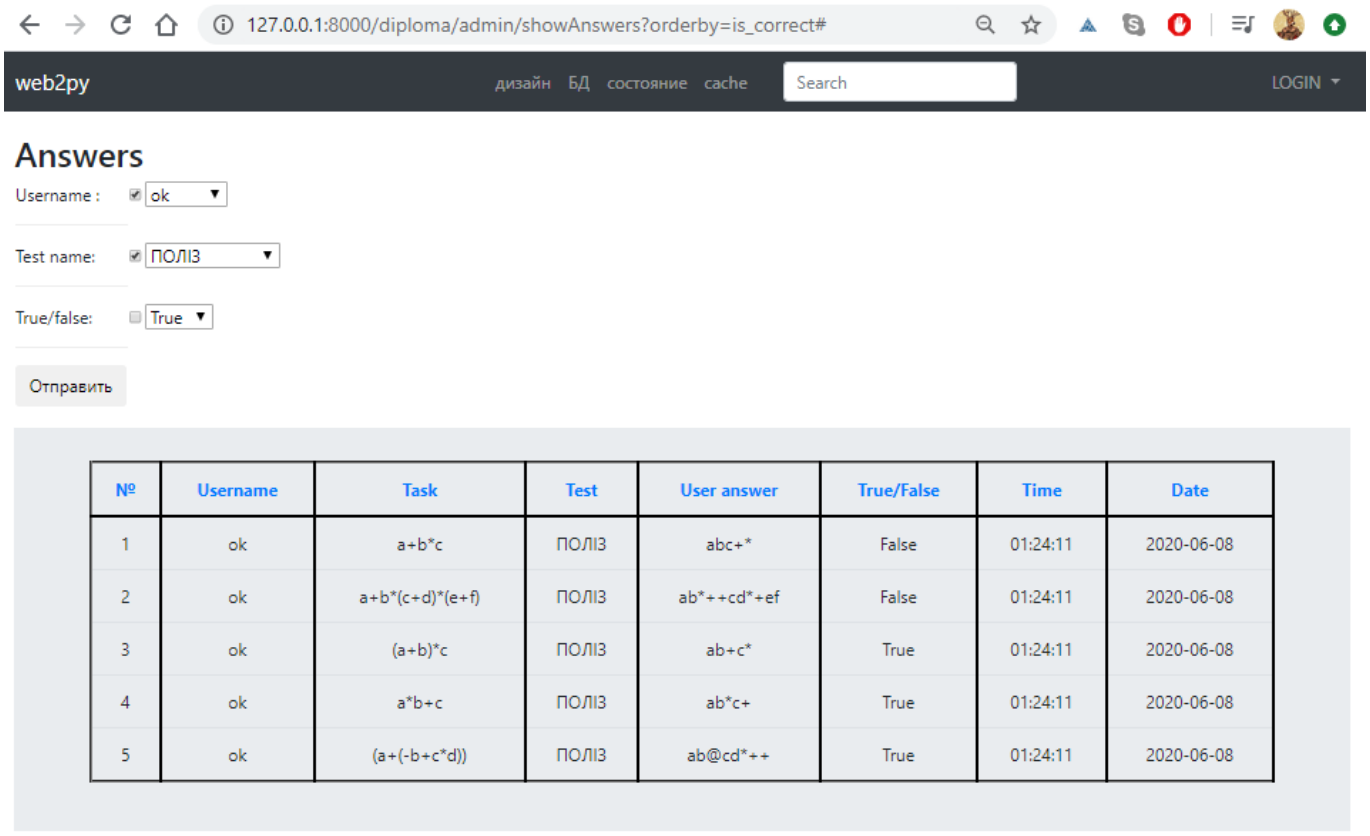
Рисунок 5.18 – Демонстрація роботи фільтрації даних

### 5.4.2 Алгоритм сортування даних історії відповідей

Система надає можливість адміністратору сортувати вибірку даних за одним з наступних параметрів: за логіном користувача, за завданням тесту, за назвою тесту, за відповіддю користувача, за правильністю відповіді, за часом та датою.

Сортування відбувається за зростанням. Тобто, якщо тип даних поля Boolean, то спочатку виводимуться строки зі значенням поля False, а вже потім строки зі значенням поля True, оскільки логічне значення False = 0, а True >= 1. Якщо дані формату строки тесту, то рядки виводимуться за алфавітним порядком. Якщо дані типу часу або дати, то виводимуться спочатку рядки з пізнішою датою або меншим значенням часу.

На Рисунку 5.19 дані відсортовано за правильністю відповіді:




| № | Username | Task | Test | User answer | True/False | Time | Date |
|---|---|---|---|---|---|---|---|
| 1 | ok | a+b*c | ПОЛІЗ | abc+* | False | 01:24:11 | 2020-06-08 |
| 2 | ok | a+b*(c+d)*(e+f) | ПОЛІЗ | ab*+cd*+ef | False | 01:24:11 | 2020-06-08 |
| 3 | ok | (a+b)*c | ПОЛІЗ | ab+c* | True | 01:24:11 | 2020-06-08 |
| 4 | ok | a*b+c | ПОЛІЗ | ab*c+ | True | 01:24:11 | 2020-06-08 |
| 5 | ok | (a+(-b+c*d)) | ПОЛІЗ | ab@cd*++ | True | 01:24:11 | 2020-06-08 |

Рисунок 5.19 – Демонстрація роботи сортування даних

# ВИСНОВКИ

У ході виконання даної роботи було розвинено розуміння і навички проектування, розробки баз даних, графічних інтерфейсів, концептуальної, було вивчено предметну область «Автоматизована система тестування», спроектовано базу даних та розроблено інформаційну систему web-сайту з тестування студентів. Для роботи з системою необхідно лише доступ до мережі Internet.

Було вирішено наступні поставлені задачі:

1. Впроваджено систему аутентифікації користувачів системи з можливістю реєстрації.
2. Розроблено та запрограмовано алгорим проходження тестування користувачем. Користувачу надається можливість обирати потрібний тест, а після його проходження надається інформація про правильність виконання завдань.
3. Розроблено та запрограмовано алгоритм взаємодії адміністратора системи з базою даних шляхом внесення змін в таблиці, які відповідають за наповнення системи переліком тестів та завданнями тестів. Адміністратор може редагувати як зміст завдання кожного тесту, так і редагувати перелік тестів та їх назви.
4. Розроблено та запрограмовано алгоритм взаємодії адміністратора системи з користувацькою історією відповідей на завдання тестів. Системою надається можливість перегляду повного списку відповідей, або фільтрувати отримані дані та сортувати їх. Фільтрація даних відбувається за параметрами логіну користувача, назви тесту або правильності відповіді. Сортування даних відбувається за зростанням числових даних або за алфавітним порядком.
5. Запрограмовано зручний інтуїтивно зрозумілий інтерфейс як для системи тестування, так і для системи адміністрування.

В процесі розробки програмного забезпечення були використані такі програмні продукти як додаток web2py.exe, Google Chome, DB Browser, Visual Studio Code.

Програмний код було написано мовою Python з використанням фреймворку web2py, методів класу pyDAL. Код для відображення веб-сторінок сайту написано мовою програмування HTML з використанням таблиці стилів CSS.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Di Pierro M. WEB2PY COMPLETE REFERENCE MANUAL, 5TH EDITION [Електронний ресурс] / Massimo Di Pierro – Режим доступу до ресурсу: http://www.web2py.com/books/default

2. SQLite. About SQLite [Електронний ресурс] / SQLite – Режим доступу до ресурсу: https://www.sqlite.org/about.html.

3. Рогачев С. Обобщённый Model-View-Controller [Електронний ресурс] / Сергей Рогачев – Режим доступу до ресурсу: http://rsdn.org/article/patterns/generic-mvc.xml

4. Буч Г., Рамбо Д., Якобсон И. Язык UML. Руководство пользователя / Гради Буч, Джеймс Рамбо, Ивар Якобсон., 2007.

5. Бьюли А. Изучаем SQL / Алан Бьюли., 2007.

6. Сегеда І. В. Системи баз даних: Комп’ютерний практикум / І. В. Сегеда, О. А. Дацюк. – Київ: КПІ ім. Ігоря Сікорського, 2019. – 43 с.

# ДОДАТОК А

Автоматизована система тестування студентів

Специфікація

УКР.НТУУ”КПІ імені Ігоря Сікорського”_ТЕФ_АПЕПС_ТР-62135_20Б

Аркушів 1

Київ 2020

| Позначення | Найменування | Примітки |
|---|---|---|
| Документація | | |
| УКР.НТУУ”КПІ ”_ТЕФ_АПЕПС_ ТР-62135_20Б | Записка.docx | Текстова частина дипломної роботи |
| Компоненти | | |
| УКР.НТУУ”КПІ ”_ТЕФ_АПЕПС_ ТР-62135_20Б 12-1 | default.py | Файл компоненту Controller що визначає алгоритм взаємодії з інтерфейсом користувача |
| УКР.НТУУ”КПІ ”_ТЕФ_АПЕПС_ ТР-62135_20Б 12-2 | admin.py | Файл компоненту Controller що визначає алгоритм взаємодії з інтерфейсом адміністратора |
| УКР.НТУУ”КПІ ”_ТЕФ_АПЕПС_ ТР-62135_20Б 12-3 | layout.html | Файл компоненту View, що визначає розмітку заголовку та підпису веб-сторінок |
| УКР.НТУУ”КПІ ”_ТЕФ_АПЕПС_ ТР-62135_20Б 12-4 | default.html | Файл компоненту View, що визначає розмітку веб-сторінок інтерфейсу користувача |
| УКР.НТУУ”КПІ ”_ТЕФ_АПЕПС_ ТР-62135_20Б 12-5 | admin.html | Файл компоненту View, що визначає розмітку веб-сторінок інтерфейсу адміністратора |
| УКР.НТУУ”КПІ ”_ТЕФ_АПЕПС_ ТР-62135_20Б 12-6 | editor_test.html | Файл для написання студентом граматик |
| УКР.НТУУ”КПІ ”_ТЕФ_АПЕПС_ ТР-62135_20Б 12-7 | db.py | Файл для збереження програмного коду бази даних |

# ДОДАТОК Б

Автоматизована система тестування студентів

Текст програми

УКР.НТУУ”КПІ імені Ігоря Сікорського”_ТЕФ_АПЕПС_ ТР-62135_20Б 12-1

Аркушів 11

Київ 2020

## default.py

```
# ---------------------------------------------------------------------
# SITE
# ---------------------------------------------------------------------
import os
import socket
import datetime
import copy
import gluon.contenttype
import gluon.fileutils
from gluon._compat import iteritems
from datetime import date
from datetime import datetime

#-----------CHOOSING THE REQUIERED TEST----------------
@auth.requires_login()
def index():
    response.view = 'default.html'
    rows = db.executesql('SELECT * FROM auth_event')
    row = rows[0]

    today = date.today()
    time = datetime.now().time().strftime("%H:%M:%S")
    response.flash = time
    return dict()

#-----------SELECTION OF THE CHOOSED TABLE--------------
def select():
    if not session.counter:
        session.counter = 1
    else:
        session.counter += 1
    curr_id = auth.user.id

    authes = db.executesql('SELECT * FROM auth_event ORDER BY id DESC LIMIT 1')
    authEvent = authes[0]
    arr = []
    names = []
    testname = request.args[0]
    rows = db.executesql("SELECT * FROM tasks WHERE name=?", [testname])

    counter = 0
    for row in rows:
        counter = counter + 1
        names.append(row[0])
        arr.extend([counter, ". ", row[2], ". Answer: ", INPUT(_type="textbox", _name = str(row[0])), HR()])

    arr.append(INPUT(_type="submit"))
    form = FORM(arr)

    if form.accepts(request, session, keepvalues = True):
        today = date.today()
        nowtime = datetime.now().time().strftime("%H:%M:%S")

        for i in names:
            ansRows = db.executesql('SELECT * FROM tasks WHERE id=?', [i])
            ansRow = ansRows[0]
            isCorrect = True
            if form.vars[str(i)] == ansRow[3]:
               isCorrect = True
            else:
               isCorrect = False
            db.userAnswer.insert(id_user=curr_id, num_session=session.counter, id_authEvent=authEvent[0],
id_expression=i, name_table_expression=testname, answer=form.vars[str(i)], is_correct = isCorrect, date = today,
time = nowtime)

        redirect(URL('result', args = [testname]))

    response.view = 'default.html'
    return dict(form=form)

def result():
    response.view = 'default.html'
    return dict()
```

```
def adminPanel():
    return dict()

def showTestList():
    tables = db().select(db.table.ALL)
    arr = []
    for table in tables:
        arr.extend([table.name, " ", INPUT(_type="button", name="Modify")])

    form = FORM(arr)

    return dict(form=form)

def user():
    return dict(form=auth())
```

## admin.py

```
# -*- coding: utf-8 -*-
import unicodedata
import os
import socket
import datetime
import copy
import gluon.contenttype
import gluon.fileutils
from gluon._compat import iteritems

is_gae = request.env.web2py_runtime_gae or False

global_env = copy.copy(globals())
global_env['datetime'] = datetime

response.view = 'admin.html'
menu = True

if menu:
    response.menu = [[T('design'), False, URL('admin', 'default', 'design',
                 args=[request.application])], [T('db'), False,
                 URL('index')], [T('state'), False,
                 URL('state')], [T('cache'), False,
                 URL('ccache')]]

if False and request.tickets_db:
    from gluon.restricted import TicketStorage
    ts = TicketStorage()
    ts._get_table(request.tickets_db, ts.tablename, request.application)

def get_databases(request):
    dbs = {}
    for (key, value) in global_env.items():
        try:
            cond = isinstance(value, GQLDB)
        except:
            cond = isinstance(value, SQLDB)
        if cond:
            dbs[key] = value
    return dbs

databases = get_databases(None)

def deleteTable():
    #db.executesql('DROP TABLE IF EXISTS deleteMe')
    #db.deleteMe.drop()
    #db.executesql('DELETE * FROM deleteMe')
    db.deleteMe.insert(name='me')
    redirect(URL('index'))
    return dict()


# ----------------------------------------------------------------------
# MY METHODS
# ----------------------------------------------------------------------
```

```
#-----------------------------MAIN MENU----------------------------------
def index():
    return dict()


# ---------------------------------------------------------------------
#-----------------------------TEST LIST----------------------------------
# ---------------------------------------------------------------------
def showTests():
    return dict(databases=databases)

def addTest():
    if request.vars.flag:
        response.flash = 'Test created!'

    arr = []

    arr.extend(["Test name", ": ", INPUT(_type="textbox", _required=True, _name = str("name")), HR()])
    arr.append(INPUT(_type="submit"))
    arr.append(" ")

    form = FORM(arr)
    form.add_button('Back', URL('showTests'))

    if form.accepts(request.vars, session, keepvalues = True):
        db.tests.insert(name = form.vars["name"])
        redirect(URL('addTest', vars=dict(flag=True)))

    return dict(form=form)

def editTest():
    arr = []
    testname = request.args[0]
    testid = request.vars.id

    rows = db.executesql('SELECT * FROM tests WHERE id=?', [testid])
    row = rows[0]

    arr.extend(["Test name", ": ", INPUT(_type="textbox", _required=True, _name = str("name"), _placeholder=row[1]),
HR()])
    arr.append(INPUT(_type="submit", _onclick="return confirm('Are you sure?')"))
    arr.append(" ")

    form = FORM(arr)
    form.add_button('Back', URL('showTests'))

    if form.accepts(request.vars, session, keepvalues = True):
        db(db.tests.id == testid).update(name = form.vars["name"])
        db(db.tasks.name == testname).update(name = form.vars["name"])
        db(db.userAnswer.name_table_expression == testname).update(name_table_expression = form.vars["name"])
        redirect(URL('showTests'))

    return dict(form=form)

def deleteTest():
    testname = request.args[0]
    testid = request.vars.id
    db.executesql('DELETE FROM tests WHERE id=?', [testid])
    db.executesql('DELETE FROM tasks WHERE name=?', [testname])

    redirect(URL('showTests'))
    return dict()


# ---------------------------------------------------------------------
#-----------------------------TASK LIST----------------------------------
# ---------------------------------------------------------------------
def select():
    return dict()

def addTask():
    if request.vars.flag:
        response.flash = 'Task added!'

    arr = []
    testname = request.args[0]
```

```
    arr.extend(["Expression", ": ", INPUT(_type="textbox", _required=True, _name = str("expression")), HR()])
    arr.extend(["Answer", ": ", INPUT(_type="textbox", _required=True, _name = str("answer")), HR()])
    arr.append(INPUT(_type="submit"))
    arr.append(" ")

    form = FORM(arr)

    if request.vars.testList:
        form.add_button('Back', URL('showTests'))
    else:
        form.add_button('Back', URL('select', args = [testname]))


    if form.accepts(request.vars, session, keepvalues = True):
        db.tasks.insert(name = testname, expression = form.vars["expression"], answer = form.vars["answer"])

        if request.vars.testList:
            redirect(URL('addTask', args = [testname], vars=dict(flag=True, testList=True)))
        else:
            redirect(URL('addTask', args = [testname], vars=dict(flag=True)))

    return dict(form=form, table=db.tasks)

def editTask():
    arr = []
    testname = request.args[0]
    testid = request.vars.id

    rows = db.executesql('SELECT * FROM tasks WHERE id=?', [testid])
    row = rows[0]

    arr.extend(["Expression", ": ", INPUT(_type="textbox", _required=True, _name = str("expression"),
_placeholder=row[2]), HR()])
    arr.extend(["Answer", ": ", INPUT(_type="textbox", _required=True, _name = str("answer"), _placeholder=row[3]),
HR()])
    arr.append(INPUT(_type="submit", _onclick="return confirm('Are you sure?')"))
    arr.append(" ")

    form = FORM(arr)
    form.add_button('Back', URL('select', args = [testname]))

    if form.accepts(request.vars, session, keepvalues = True):
        db(db.tasks.id == testid).update(expression = form.vars["expression"], answer = form.vars["answer"])
        redirect(URL('select', args = [testname]))

    return dict(form=form)

def deleteTask():
    testname = request.args[0]
    testid = request.vars.id

    db.executesql('DELETE FROM tasks WHERE id=?', [testid])
    db.executesql('DELETE FROM userAnswer WHERE id_expression=?', [testid])
    redirect(URL('select', args = [testname]))
    return dict()


# ---------------------------------------------------------------------
#-----------------------------ANSWER LIST-------------------------------
# ---------------------------------------------------------------------


def showAnswers():


    rowsName = db.executesql('SELECT * FROM auth_user')
    arrName = []
    for row in rowsName:
        arrName.append(row[8])

    rowsTest = db.executesql('SELECT * FROM tests')
    arrTest = []
    for row in rowsTest:
        arrTest.append(row[1])

    form = FORM(TABLE(TR("Username : ", INPUT(_type="checkbox", _name = str("nameFilter")), SELECT(arrName,
_name="nameList")),
```

```
                    TR(HR()),
                    TR("Test name: ", INPUT(_type="checkbox", _name = str("testFilter")), SELECT(arrTest,
_name="testList")),
                    TR(HR()),
                    TR("True/false: ", INPUT(_type="checkbox", _name = str("iscorrectFilter")), SELECT(["True",
"False"], _name="iscorrectList")),
                    TR(HR()),
                    TR(INPUT(_type="submit"))))


    requestsql = 'SELECT * FROM userAnswer WHERE id>0'


    if form.accepts(request.vars, session, keepvalues = True):
        if form.vars["nameFilter"]:
            tmps = db.executesql('SELECT * FROM auth_user WHERE username=?', [form.vars["nameList"]])
            tmp = tmps[0]
            iduser = tmp[0]
            requestsql = requestsql + (' AND id_user=' + str(iduser))
        if form.vars["testFilter"]:
            requestsql = requestsql + (' AND name_table_expression="' + form.vars["testList"] + '"')
        if form.vars["iscorrectFilter"]:
            requestsql = requestsql + (' AND is_correct="' + form.vars["iscorrectList"] + '"')

    if request.vars.orderby:
        if request.vars.orderby == "id":
            requestsql = requestsql + (' ORDER BY id')
        elif request.vars.orderby == "id_user":
            requestsql = requestsql + (' ORDER BY id_user')
        elif request.vars.orderby == "id_expression":
            requestsql = requestsql + (' ORDER BY id_expression')
        elif request.vars.orderby == "name_table_expression":
            requestsql = requestsql + (' ORDER BY name_table_expression')
        elif request.vars.orderby == "answer":
            requestsql = requestsql + (' ORDER BY answer')
        elif request.vars.orderby == "is_correct":
            requestsql = requestsql + (' ORDER BY is_correct')
        elif request.vars.orderby == "time":
            requestsql = requestsql + (' ORDER BY time')
        elif request.vars.orderby == "date":
            requestsql = requestsql + (' ORDER BY date')

    answers = db.executesql(requestsql)

    return dict(form=form, answers=answers)
```

## default.html

```
{{extend 'layout.html'}}
{{import unicodedata}}
{{if request.function=='index':}}

    <h2>{{=T("SELECT TEST")}}</h2>
    <div class="jumbotron jumbotron-fluid" style="padding:30px;word-wrap:break-word;">
        <div class="container center">
            <table class="table">
            {{tests = db.executesql('SELECT * FROM tests')}}
            {{counter = 0}}
            {{for test in tests:}}

                {{if counter == 3:}}
                    <tr></tr>
                    {{counter = 0}}
                {{pass}}
                {{qry = '%s.%s.id>0'%('db', db.tests)}}
                <th>{{=A("%s" % (test[1]), _href = URL('select', args=[test[1]], vars=dict(query=qry)), _class =
"btn btn-secondary")}}</th>
                {{counter = counter + 1}}
            {{pass}}
            </table>
        </div>
    </div>
{{elif request.function=='select':}}
    <h2>{{=T(request.args[0])}}</h2>
    {{=form}}
```

```
{{elif request.function=='result':}}
    <h2>{{=T("RESULT " + request.args[0])}}</h2>
    <div class="jumbotron jumbotron-fluid" style="padding:30px;word-wrap:break-word;">
    {{curr_id = auth.user.id}}
    {{rows = db.executesql('SELECT * FROM auth_event WHERE user_id=? ORDER BY id DESC LIMIT 1', [curr_id])}}
    {{rowAuth_event = rows[0]}}
    {{form = rowAuth_event[0]}}
    {{rows = db.executesql('SELECT * FROM userAnswer WHERE id_authEvent=? AND num_session=?', (rowAuth_event[0],
session.counter))}}

    {{counter = 0}}
    <table>
    {{for row in rows:}}
        {{counter = counter + 1}}
        {{ansRows = db.executesql('SELECT * FROM tasks WHERE id=?', [row[4]])}}
        {{ansRow = ansRows[0]}}

        {{if row[6] == ansRow[3]:}}
            <tr>
                <td style="padding-right:50px;">{{=B(counter, ". ", 'Expression: ', ansRow[2])}}</td>
                <td>{{=B('User`s answer: ', row[6],  ' - Correct')}}</td>
            </tr>
        <tr><td>{{=B(HR())}}</td></tr>
        {{else:}}
            <tr>
                <td style="padding-right:50px;">{{=B(counter, ". ", 'Expression: ', ansRow[2])}}</td>
                <td>{{=B('User`s answer: ', row[6],  ' - Incorrect')}}</td>
            </tr>
            <tr><td>{{=B(HR())}}</td></tr>
        {{pass}}

    {{pass}}
    </table>
    {{=A('Go home', _href = URL('index'), _class = "btn btn-secondary")}}
    </div>
{{pass}}
```

## admin.html

```
{{extend 'layout.html'}}
{{import unicodedata}}


{{if request.function=='index':}}
<h2>{{=T("Administrator`s panel")}}</h2>
<div class="jumbotron jumbotron-fluid" style="word-wrap:break-word;">
    <div class="container center">
        <a class="btn btn-secondary" href="{{=URL('showTests')}}">{{=T('Tests')}}</a>
        <a class="btn btn-secondary" href="{{=URL('showAnswers')}}">{{=T('Answers')}}</a>
    </div>
</div>

{{elif request.function=='showTests':}}
<h2>{{=T("Tests")}}</h2>
<div class="jumbotron jumbotron-fluid" style="padding:30px;word-wrap:break-word;">
    {{=A(str(T("Add new test")), _href = URL('addTest'), _class = "btn btn-secondary")}} 
    {{=A(str(T("Back")), _href = URL('index'), _class = "btn btn-secondary")}}
    <div class="container center">
        <table class="table">
            {{tests = db.executesql('SELECT * FROM tests')}}
            {{counter = 0}}
            {{for test in tests:}}
                {{qry = '%s.%s.id>0'%('db', db.tests)}}
                <tr>
                    <th style="font-size: 1.5em; width: 1200px;">
                        {{=A("%s" % (test[1]), _href = URL('select', args=[test[1]], vars=dict(query=qry)))}}
                    </th>
                    <td>
                        {{=A(str(T('Add new task')), _href = URL('addTask', args = [test[1]], vars =
dict(testList=True)), _class = "btn btn-secondary")}}
                    </td>
                    <td>
                        {{=A(str(T('Rename test')), _href = URL('editTest', args = [test[1]], vars =
dict(id=test[0])), _class = "btn btn-secondary")}}
                    </td>
```

```
                    <td>
                        {{=A(str(T('Delete test')), _href = URL('deleteTest', args = [test[1]], vars =
dict(id=test[0])), _class = "btn btn-secondary", _onclick="return confirm('Are you sure?')")}}
                    </td>
                </tr>
            {{pass}}
        </table>
    </div>
</div>

{{elif request.function=='addTest':}}
    <h2>{{=T("Add new test")}}</h2>
    <hr>
    {{=form}}

{{elif request.function=='editTest':}}
    <h2>{{=T("Rename test " + request.args[0])}}</h2>
    {{=form}}




{{elif request.function=='select':}}
<h2>{{=T(request.args[0])}}</h2>
<div class="jumbotron jumbotron-fluid" style="padding:30px; word-wrap:break-word;">
    {{=A(str(T("Add new task")), _href = URL('addTask', args = [request.args[0]]), _class = "btn btn-
secondary")}} 
    {{=A(str(T("Back")), _href = URL('showTests'), _class = "btn btn-secondary")}}
    {{=B(HR())}}
    <div class="container center">
        <table class="table" width="100%" border="3" cellpadding="0">
            {{testname = request.args[0]}}
            {{rows = db.executesql('SELECT * FROM tasks WHERE name=?', [testname])}}

            {{counter = 0}}

            <tr align="center">
                    <td align = "center" valign="center" style="border-bottom: 3px solid black; border-right: 3px
solid black;">{{=B("№")}}</td>
                    <td align = "center" valign="center" style="border-bottom: 3px solid black; border-right: 3px
solid black;">{{=B("Expression")}}</td>
                    <td align = "center" valign="center" style="border-bottom: 3px solid black; border-right: 3px
solid black;">{{=B("Answer")}}</td>
                    <td align = "center" valign="center" style="border-bottom: 3px solid black;"> </td>
            </tr>
            {{for row in rows:}}
                {{counter = counter + 1}}
                <tr align="center">
                    <td align = "center" valign="center" style="border-right: 3px solid black;">{{=T("%s" %
(counter))}}</td>
                    <td align = "center" valign="center" style="border-right: 3px solid black;">{{=T("%s" %
(row[2]))}}</td>
                    <td align = "center" valign="center" style="border-right: 3px solid black;">{{=T("%s" %
(row[3]))}}</td>
                    <td>
                        {{=A(str(T("Edit task")), _href = URL('editTask', args = [request.args[0]], vars =
dict(id=row[0])), _class = "btn btn-secondary")}} 
                        {{=A(str(T("Delete task")), _href = URL('deleteTask', args = [request.args[0]], vars =
dict(id=row[0])), _class = "btn btn-secondary", _onclick="return confirm('Are you sure?')")}}
                    </td>
                </tr>
            {{pass}}
        </table>
    </div>
</div>

{{elif request.function=='addTask':}}
    <h2>{{=T("Add new task to " + request.args[0])}}</h2>
    <hr>
    {{=form}}

{{elif request.function=='editTask':}}
    <h2>{{=T("Edit task from " + request.args[0])}}</h2>
    {{=form}}
```

```
{{elif request.function=='showAnswers':}}
    <h2>{{=T("Answers")}}</h2>
    {{=form}}
    <div class="jumbotron jumbotron-fluid" style="padding:30px; margin-top:20px; word-wrap:break-word;">
        <div class="container center">
            <table class="table" border="3">
                {{counter = 0}}
                {{rows=answers}}
                <tr>
                    {{url = None}}

                    <td align = "center" valign="center" style="border-bottom: 3px solid black; border-right: 3px
solid black;">
                        {{if request.vars.orderby=="id":}}
                            {{url = URL('showAnswers')}}
                        {{elif request.vars.orderby in [None, "id_user", "id_expression", "name_table_expression",
"answer", "is_correct", "time", "date"]:}}
                            {{url = URL('showAnswers', vars=dict(orderby="id"))}}
                        {{pass}}
                        <b>{{=A(str(T('№')), _href = url)}}</b>
                    </td>
                    <td align = "center" valign="center" style="border-bottom: 3px solid black; border-right: 3px
solid black;">
                        {{if request.vars.orderby=="id_user":}}
                            {{url = URL('showAnswers')}}
                        {{elif request.vars.orderby in [None, "id", "id_expression", "name_table_expression",
"answer", "is_correct", "time", "date"]:}}
                            {{url = URL('showAnswers', vars=dict(orderby="id_user"))}}
                        {{pass}}
                        <b>{{=A(str(T('Username')), _href = url)}}</b>
                    </td>
                    <td align = "center" valign="center" style="border-bottom: 3px solid black; border-right: 3px
solid black;">
                        {{if request.vars.orderby=="id_expression":}}
                            {{url = URL('showAnswers')}}
                        {{elif request.vars.orderby in [None, "id", "id_user", "name_table_expression", "answer",
"is_correct", "time", "date"]:}}
                            {{url = URL('showAnswers', vars=dict(orderby="id_expression"))}}
                        {{pass}}
                        <b>{{=A(str(T('Task')), _href = url)}}</b>
                    </td>
                    <td align = "center" valign="center" style="border-bottom: 3px solid black; border-right: 3px
solid black;">
                        {{if request.vars.orderby=="name_table_expression":}}
                            {{url = URL('showAnswers')}}
                        {{elif request.vars.orderby in [None, "id", "id_user", "id_expression", "answer",
"is_correct", "time", "date"]:}}
                            {{url = URL('showAnswers', vars=dict(orderby="name_table_expression"))}}
                        {{pass}}
                        <b>{{=A(str(T('Test')), _href = url)}}</b>
                    </td>
                    <td align = "center" valign="center" style="border-bottom: 3px solid black; border-right: 3px
solid black;">
                        {{if request.vars.orderby=="answer":}}
                            {{url = URL('showAnswers')}}
                        {{elif request.vars.orderby in [None, "id", "id_user", "id_expression",
"name_table_expression", "is_correct", "time", "date"]:}}
                            {{url = URL('showAnswers', vars=dict(orderby="answer"))}}
                        {{pass}}
                        <b>{{=A(str(T('User answer')), _href = url)}}</b>
                    </td>
                    <td align = "center" valign="center" style="border-bottom: 3px solid black; border-right: 3px
solid black;">
                        {{if request.vars.orderby=="is_correct":}}
                            {{url = URL('showAnswers')}}
                        {{elif request.vars.orderby in [None, "id", "id_user", "id_expression",
"name_table_expression", "answer", "time", "date"]:}}
                            {{url = URL('showAnswers', vars=dict(orderby="is_correct"))}}
                        {{pass}}
                        <b>{{=A(str(T('True/False')), _href = url)}}</b>
                    </td>
                    <td align = "center" valign="center" style="border-bottom: 3px solid black; border-right: 3px
solid black;">
                        {{if request.vars.orderby=="time":}}
                            {{url = URL('showAnswers')}}
```

```
                        {{elif request.vars.orderby in [None, "id", "id_user", "id_expression",
"name_table_expression", "answer", "is_correct", "date"]:}}
                            {{url = URL('showAnswers', vars=dict(orderby="time"))}}
                        {{pass}}
                        <b>{{=A(str(T('Time')), _href = url)}}</b>
                    </td>
                    <td align = "center" valign="center" style="border-bottom: 3px solid black; border-right: 3px
solid black;">
                        {{if request.vars.orderby=="date":}}
                            {{url = URL('showAnswers')}}
                        {{elif request.vars.orderby in [None, "id", "id_user", "id_expression",
"name_table_expression", "answer", "is_correct", "time"]:}}
                            {{url = URL('showAnswers', vars=dict(orderby="date"))}}
                        {{pass}}
                        <b>{{=A(str(T('Date')), _href = url)}}</b>
                    </td>
                </tr>
                {{for row in rows:}}
                    {{counter = counter + 1}}
                    {{userRows = db.executesql('SELECT * FROM auth_user WHERE id=?', [row[1]])}}
                    {{userRow = userRows[0]}}
                    {{taskRows = db.executesql('SELECT * FROM tasks WHERE id=?', [row[4]])}}
                    {{taskRow = taskRows[0]}}
                    <tr>
                        <td align = "center" valign="center" style="border-right: 3px solid black;">{{=T("%s" %
(counter))}}</td>
                        <td align = "center" valign="center" style="border-right: 3px solid black;">{{=T("%s" %
(userRow[8]))}}</td>
                        <td align = "center" valign="center" style="border-right: 3px solid black;">{{=T("%s" %
(taskRow[2]))}}</td>
                        <td align = "center" valign="center" style="border-right: 3px solid black;">{{=T("%s" %
(row[5]))}}</td>
                        <td align = "center" valign="center" style="border-right: 3px solid black;">{{=T("%s" %
(row[6]))}}</td>
                        <td align = "center" valign="center" style="border-right: 3px solid black;">{{=T("%s" %
(row[7]))}}</td>
                        <td align = "center" valign="center" style="border-right: 3px solid black;">{{=T("%s" %
(row[8]))}}</td>
                        <td align = "center" valign="center" style="border-right: 3px solid black;">{{=T("%s" %
(row[9]))}}</td>
                    </tr>
                {{pass}}
            </table>
        </div>
    </div>

{{pass}}
```

## editor_copy.html

```
<html lang="en">
{{extend 'layout.html'}}

<head>
  <style type="text/css" media="screen">
    body {
        overflow: hidden;
    }

    #editor {
                position: absolute;
        width: 650px;
                height: 700px;
    }
  </style>
</head>
<body>

<h2>{{=T(db.executesql('SELECT * FROM tests WHERE id=?', [request.vars.id_test])[0][1])}}</h2>
<hr>


{{=A(str(T('Send')), _href=URL('index'), _class="btn btn-secondary", _onclick="sendPost(editor.getValue())")}}
{{=A(str(T('Back')), _href=URL('index'), _class="btn btn-secondary")}}
```

```
        <pre id="editor">

        </pre>


        <script src="http://localhost:8000/diploma/static/src-noconflict/ace.js"  type="text/javascript" charset="utf-
8"></script>
        <script>
            var myTheme = 'deamveawer'
                var myLang = 'haskell'
            //console.log(src)
            //console.log(' before: var editor='+editor)
            var editor = ace.edit("editor");
            //console.log(' after: var editor='+editor)
            editor.setTheme("ace/theme/"+myTheme);
            editor.session.setMode("ace/mode/"+myLang);
                editor.setFontSize(14);

            function sendPost(grammar) {

                var id_task = decodeURIComponent(location.search.substr(1)).split('&');
                var id_test = decodeURIComponent(location.search.substr(2)).split('&');
                var num_session = decodeURIComponent(location.search.substr(8)).split('&');


                var params = {};
                var query = location.search.substr(1);
                var vars = query.split('&');

                for(var i = 0; i < vars.length; i++) {
                    var pair = vars[i].split('=');
                    params[pair[0]] = decodeURIComponent(pair[1]);
                }


                var formData = new FormData();
                formData.append("grammar", grammar);
                formData.append("id_task", params["id_task"]);
                formData.append("id_test", params["id_test"]);
                formData.append("num_session", params["num_session"]);

                alert(grammar)
                var xhr = new XMLHttpRequest();
                xhr.open("POST", "/diploma/default/editor_copy");
                xhr.send(formData)

            }

        </script>

        </body>
        </html>
```

## db.py

```
        from gluon.tools import Auth
        T.force(None)

        db = DAL("sqlite://storage.sqlite")

        auth = Auth(db)
        auth.define_tables(username=True, signature=True)
        auth.secure = True


        db.define_table('tasks',
                        Field('name'),
                        Field('expression'),
                        Field('answer'),
                        Field('weight'),
                        Field('type'))
        db.tasks.writable = False
```

```
db.define_table('tests',
                Field('name', notnull=True, unique=True),
                Field('date'))
db.tests.writable = False


db.define_table('addition',
                Field('expression'),
                Field('answer'))


db.define_table('multiplication',
                Field('expression'),
                Field('answer'))


db.define_table('userAnswer',
                Field('id_user', 'reference auth_user'),
                Field('num_session'),
                Field('id_authEvent', 'reference auth_event'),
                Field('id_expression', 'reference tasks'),
                Field('name_table_expression'),
                Field('answer'),
                Field('is_correct'),
                Field('date'),
                Field('time'))
#db.userAnswer.writable = False
```

# ДОДАТОК В

Автоматизована система тестування студентів

Опис програмного коду

УКР.НТУУ”КПІ імені Ігоря Сікорського”_ТЕФ_АПЕПС_ ТР-62135_20Б
13-1

Аркушів 3

Київ 2020

### В.1 Файл контролеру default.py

Згідно записів у таблиці бази даних “tests” відбувається пошук серед переліку тестів і знайдений тест потрапляє до форми, що пересилається до файлу представлення default.html, котрий зчитуючи її виводить на інтерфейс відповідну кількість елементів керування – кнопок, які відповідають за кожен доступний тест.

При натискання на одне з посилань відбувається перенаправлення користувача на веб-сторінку /result, і разом з тим формується пост-запит, що містить аргумент з текстовою строкою - назвою тесту, та змінною query - умовою вибору даних з таблиці бази даних, що є аргументом для методу select() класу DAL.

Використовуючи передані в пост-запиті змінні та аргументи користувацький метод select() створює форму з завданнями обраного тесту.

### В.2 Файл контролеру admin.py

Кожен фільтр містить checkbox – поле що відповідає за активність даного фільтру та список параметрів які можна застосувати до даного фільтру. Для логіну - це список усіх зареєстрованих користувачів. Для назви тесту - це список усіх створених тестів. Для правильності відповіді  - це параметри True або False.

Якщо активовано один або більше фільтрів, достатнью відправити форму щоб фільтри застосувалися до вибірки даних відповідно до обраних параметрів.

Система надає можливість адміністратору сортувати вибірку даних за одним з наступних параметрів: за логіном користувача, за завданням тесту, за назвою тесту, за відповіддю користувача, за правильністю відповіді, за часом та датою.

Сортування відбувається за зростанням. Тобто, якщо тип даних поля Boolean, то спочатку виводимуться строки зі значенням поля False, а вже потім строки зі значенням поля True, оскільки логічне значення False = 0, а True >= 1. Якщо дані формату строки тесту, то рядки виводимуться за алфавітним порядком. Якщо дані типу часу або дати, то виводимуться спочатку рядки з пізнішою датою або меншим значенням часу.

### В.3 Файл представлення default.html

Кнопка-посилання внизу веб-сторінки /result дозволяє користувачу повернутися на головну сторінку системи для подальшого вибору та проходження тестів. Після повернення користувача на головну сторінку лічільник сесій інкрементується і при другій спробі проходження тесту так само можна буде явно ідентифікувати зроблені користувачем відповіді для виведення на веб-сторінку результатів тестування.

У будь-який момент роботи з системою, користувач має змогу вийти зі свого аккаунту на сторінку аутентифікації користувача, при цьому до таблиці auth_event занесеться запис про вихід користувача з конкретним ідентифікатором з системи.

Користувач має змогу відредагувати дані свого аккаунту якщо перейде за відповідним посиланням у випадаючому вікні у заголовку будь-якої сторінки. Система перенаправляє користувача на сторінку /profile, де будується форма котра складається з доступних для модифікації полів імені, прізвища та логіну користувача. Електронну пошту змінити не можна. При натисканні на кнопку-посилання “Appy changes”, згенерується команда на оновлення запису з таблиці auth_user за допомогою методу класу DAL - update(), де аргументи це необхідні для редагування поля запису.

### В.4 Файл представлення admin.html

Панель адміністратора створено для надання можливості адміністратору системи створювати, редагувати та видаляти тести, та завдання кожного з тестів. Також адміністратор може переглядати історію відповідей користувачів, фільтрувати надані системою дані та сортувати їх.

Розмітка головної сторінки повністю будується у файлі представлення admin.html. Містить два посилання-кнопки, одне на сторінку редагування тестів, друге на сторінку перегляду історії відповідей користувачів на завдання тестів.

При переході адміністратора за посиланням для додавання нового тесту до списку, викликається метод addTest() модуля Controller з файлу admin.py. У ньому

формується форма з поля для введення тексту, кнопки-відправки форми та кнопки-посилання на попередню сторінку перегляду списку тестів.

При натисканні адмінісртатором на кнопку відправки форми зчитується введений текст та викликається метод insert(), котрий заносить новий запис до таблиці “tests” бази даних. Після цього системи повертає адміністратора на ту-ж сторінку для створення нового тесту і виводить push-повідомлення про те що тест було створено. Адміністратор може продовжити створювати нові тести або може повернутися на попередню сторінку перегляду списку тестів.